MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

ML-UNIT II

SCHOOL OF COMPUTER ENGINEERING & TECHNOLOGY

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Supervised learning techniques: Classification

Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

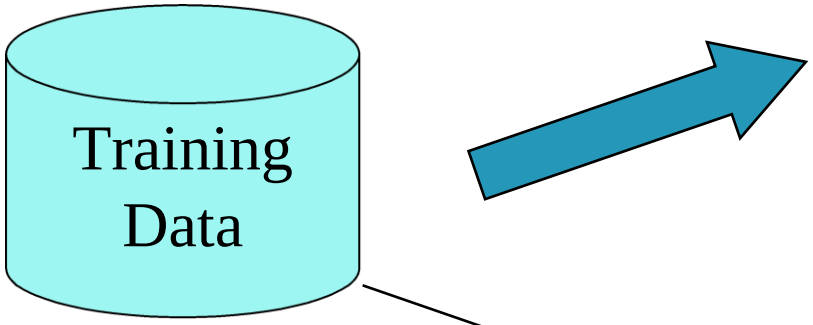
Prediction Problems: Classification vs. Numeric Prediction

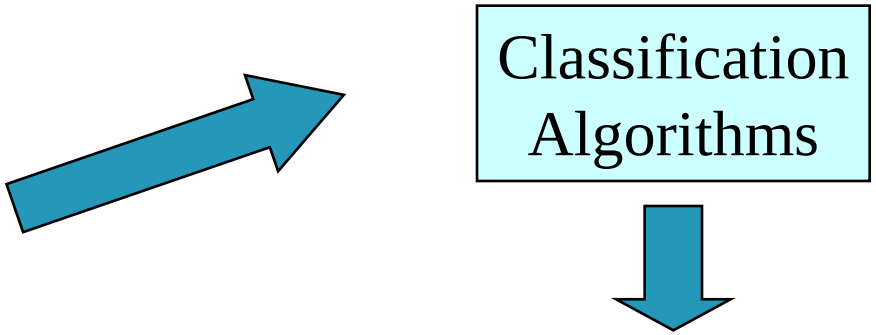
- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Numeric Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is

Classification—A Two-Step Process

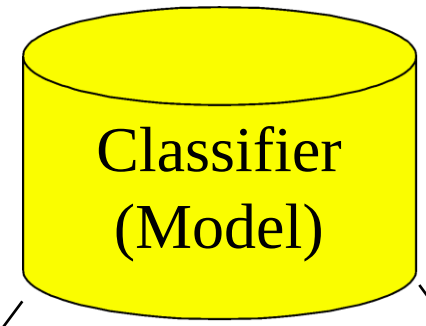
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to classify new data
- Note: If *the test set* is used to select models, it is called validation (test) set

Process (1): Model Construction



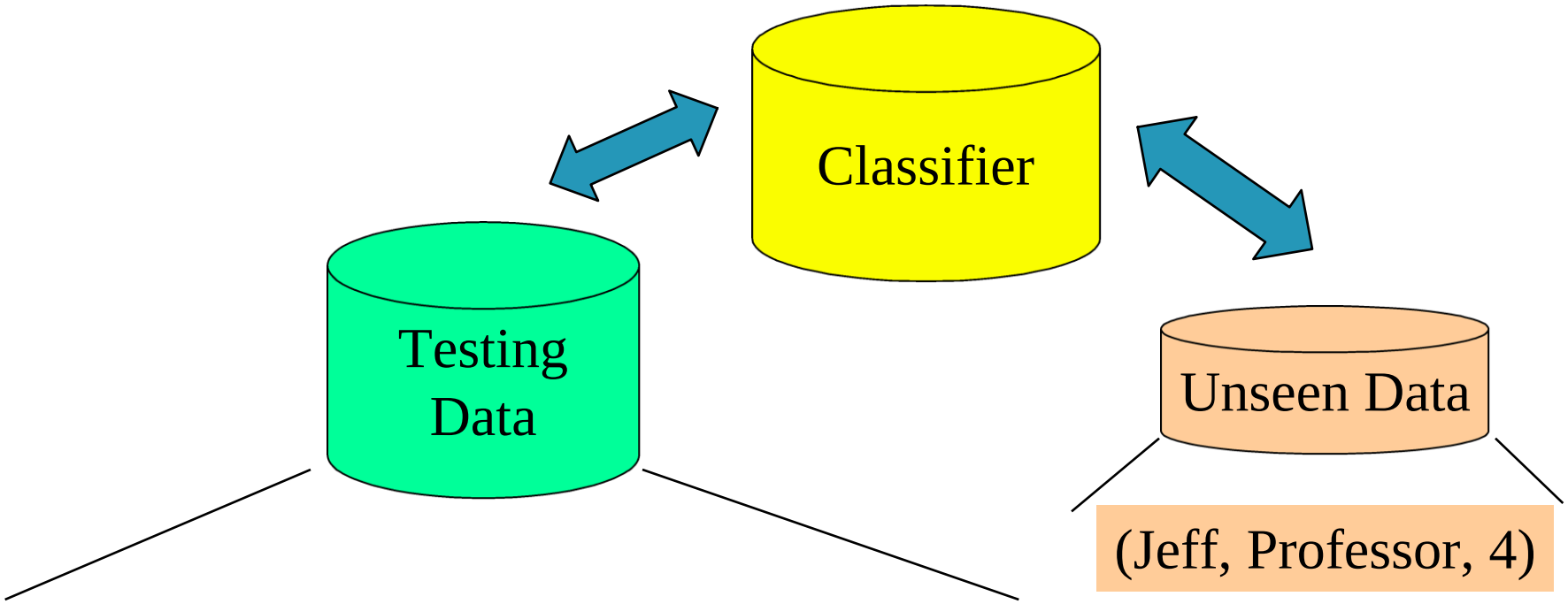


NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no



IF rank = ‘professor’
OR years > 6
THEN tenured = ‘yes’

Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

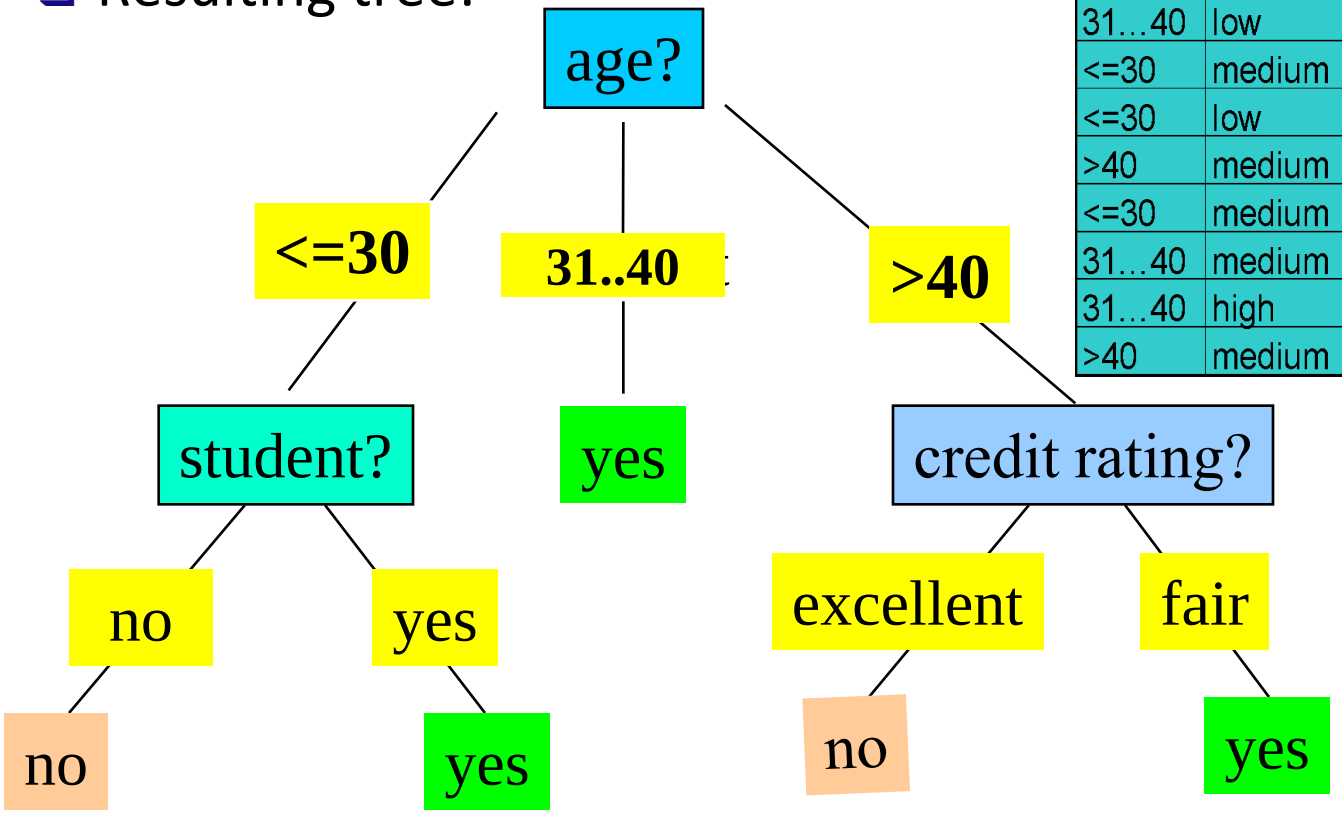
Classification: Basic Concepts

- Classification: Basic Concepts
- Decision Tree Induction

Decision Tree Induction: An Example

- Training data set: Buys_computer
- The data set follows an example of Quinlan's ID3 (Playing Tennis)
- Resulting tree:

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

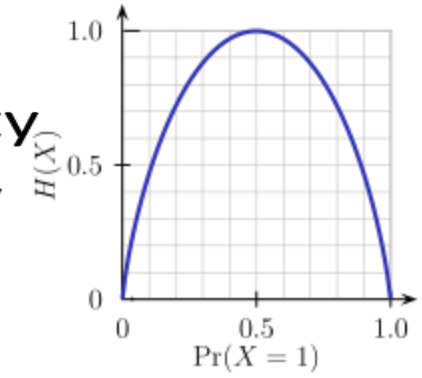


Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Brief Review of Entropy

- Entropy (Information Theory)
  - A measure of uncertainty associated with a random variable
  - Calculation: For a discrete random variable Y taking m distinct values $\{y_1, \ldots, y_m\}$,
    - $H(Y) = -\sum_{i=1}^{m} p_i \log(p_i)$, where $p_i = P(Y = y_i)$
  - Interpretation:
    - Higher entropy => higher uncertainty
    - Lower entropy => lower uncertainty
- Conditional Entropy
  - $H(Y|X) = \sum_x p(x) H(Y|X = x)$



m = 2

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection : Information gain

Class P: buys_computer = “yes”
Class N: buys_computer = “no”

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

1 : Calculate Entropy for Class Labels

$$Info(D) = -\frac{9}{14}\log_2\left(\frac{9}{14}\right) - \frac{5}{14}\log_2\left(\frac{5}{14}\right) = 0.940$$

2 : Calculate Information of Each Attribute (one by one)

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right)$$

$$= 0.694$$

3 : Calculate Gain of Each Attribute (one by one)

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

$$Gain(income) = 0.029$$

$$Gain(student) = 0.151$$

$$Gain(credit\_rating) = 0.048$$

4: Select Attribute with max gain as a root attribute

Gain(age)> any other gain. so age is selected as root node

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14}I(2,3) + \frac{4}{14}I(4,0) + \frac{5}{14}I(3,2) = 0.694$$

$\frac{5}{14}I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's.

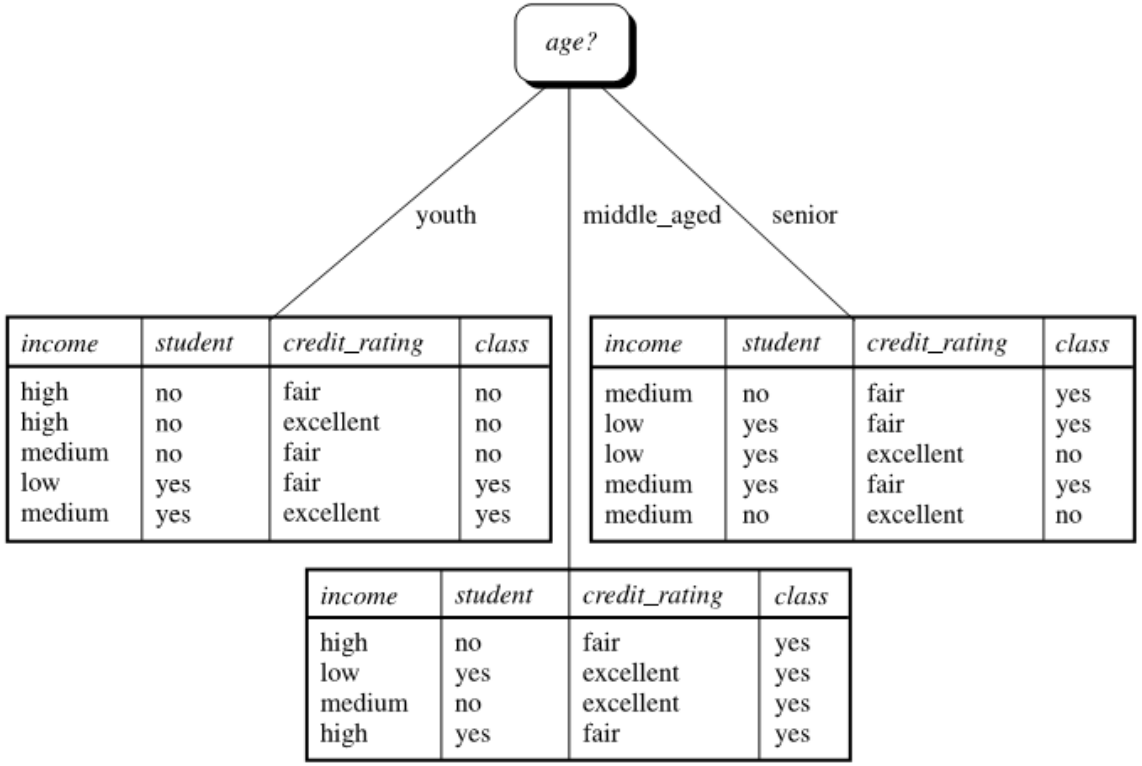
Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

Similarly,

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Intermediate DT of the buys_computer dataset



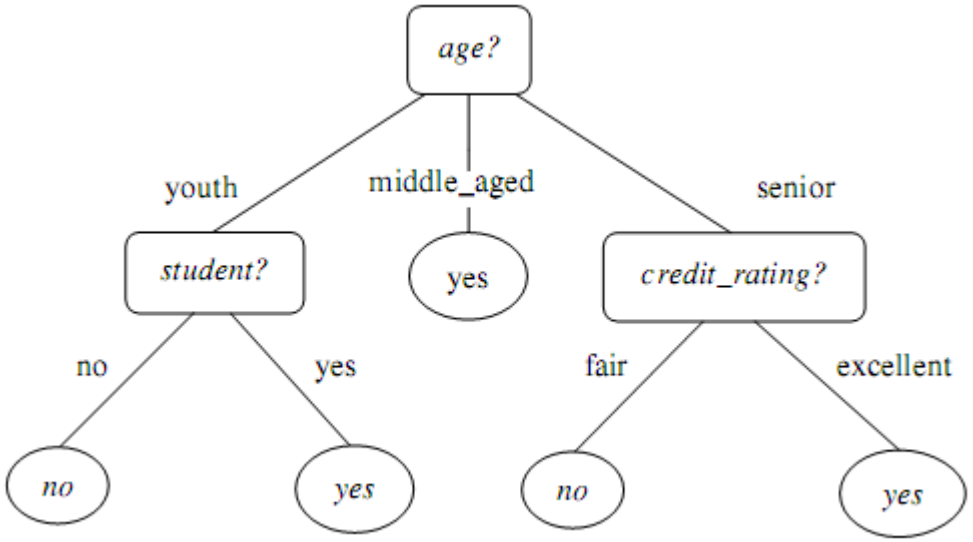
income	*student*	*credit_rating*	*class*
high	no	fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	yes

income	*student*	*credit_rating*	*class*
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

income	*student*	*credit_rating*	*class*
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Age is the splitting attribute at the root node of DT. Repeat the procedure to determine the splitting attribute(except age) along each of the branches from root node till stopping condition is reached to generate the final DT

Final DT of the buys_computer dataset



Example : DT Creation

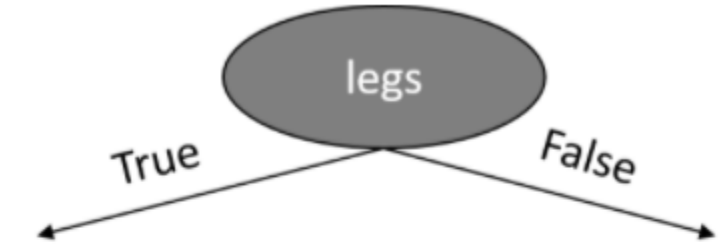
	toothed	breathes	legs	species
0	True	True	True	Mammal
1	True	True	True	Mammal
2	True	True	False	Reptile
3	False	True	True	Mammal
4	True	True	True	Mammal
5	True	True	True	Mammal
6	True	False	False	Reptile
7	True	True	False	Reptile
8	True	True	True	Mammal
9	False	True	True	Reptile

$$InfoGain(toothed) = 0.971 - 0.963547 = \mathbf{0.00745}$$

$$InfoGain(breathes) = 0.971 - 0.82647 = \mathbf{0.1445}$$

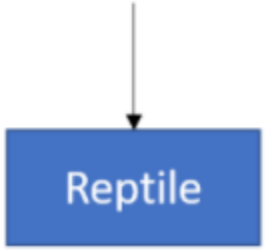
$$InfoGain(legs) = 0.971 - 0.41417 = \mathbf{0.5568}$$

Example : DT Creation



	toothed	breathes	species
0	True	True	Mammal
1	True	True	Mammal
3	False	True	Mammal
4	True	True	Mammal
5	True	True	Mammal
8	True	True	Mammal
9	False	True	Reptile

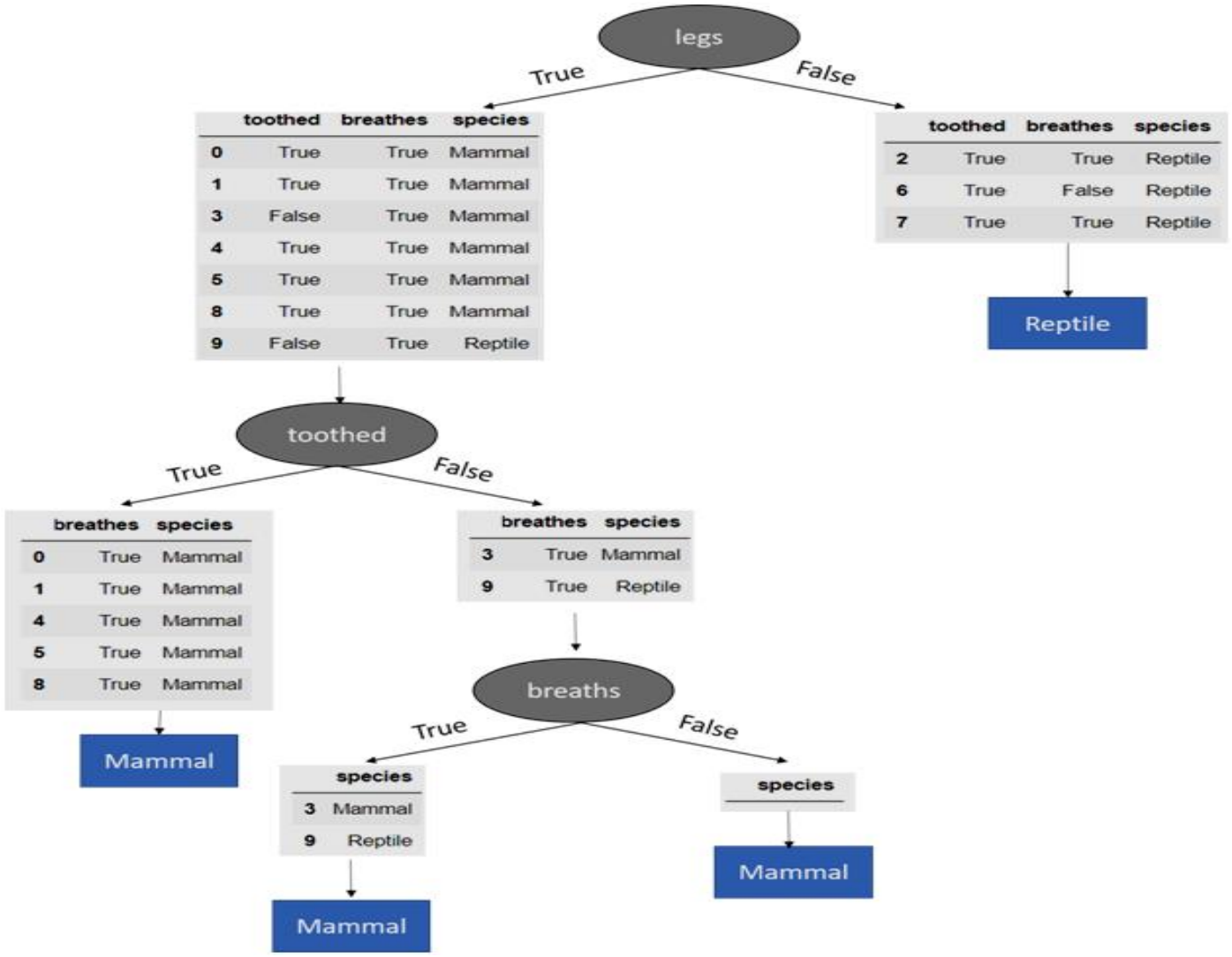
	toothed	breathes	species
2	True	True	Reptile
6	True	False	Reptile
7	True	True	Reptile



$$InfoGain(toothed) = 0.5917 - 0.285 = \mathbf{0.3067}$$

$$InfoGain(breathes) = .5917 - 0.5917 = \mathbf{0}$$

Example : DT Creation



Example : Usage of Information Gain and Entropy in DT Creation

1. Create a root node

2. Calculate the entropy of the whole (sub) dataset

3. Calculate the Information gain of each single feature and pick that feature with the larges Information gain

4. Assign the (root) node the label of the feature with the maximum information gain. Grow for each feature value an outgoing branch and add unlabelled nodes at the end

5. Split the dataset along the values of the maximum information gain feature and remove this feature from the dataset

6. For each of the sub_datasets, repeat steps 2 to 5 until a stopping criteria is satisfied →Here the recursion kicks in

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Decision Tree

- A Classifier (tree structure): used in classification and regression
- Classification mostly uses Decision tree
- Decision tree Model act as classifier : **tree structure classifier**
- New input(unlabeled, unknown) is **fed to the model** and model classifies it to a particular class
- Decision tree has two nodes:
  - **Decision** node(branch: test conducted either yes or no) [corresponds to an **Attribute**]
  - **Leaf** node(no branch) [corresponds to a **Class Label**]
- Finally leaf node is a class….(assign a class to next coming sample)

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Classification & Regression Trees (CART)

- DT creates a model that **predicts the value of a target** (or dependent variable) based on the values of several input (or independent variables)
- CART was introduced in 1984 by Leo Breiman, Jerome Friedman, Richard Olshen and Charles Stone.
- The main elements of **CART** (and any decision tree algorithm) are:
  - **Rules** for splitting data at a node based on the value of one variable
  - Stopping rules for deciding when a branch is terminal and can be split no more
  - Finally, a **prediction for the target variable** in each terminal node.

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  - GainRatio(A) = Gain(A)/SplitInfo(A)
- Ex.

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  - gain_ratio(income) = 0.029/1.557 = 0.019
- The attribute with the maximum gain ratio is selected as the splitting attribute

CART (Classification & Regression Tree)

- **C4.5 - Gain ratio:**
  - tends to prefer unbalanced splits in which one partition is much smaller than the others

- **CART (Classification & Regression Tree)**
  - CART creates a Binary Tree
  - implements Gini index as a attribute selection measure.

Gini Index (CART, IBM IntelligentMiner)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as

$$gini(D) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class *j* in *D*

- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *gini*(*D*) is defined as

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

- Reduction in Impurity:

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- The attribute provides the smallest $gini_{split}(D)$ (or the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

Computation of Gini Index

- Ex. D has 9 tuples in buys_computer = “yes” and 5 in “no”

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low, medium} and 4 in D_2

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right)$$

$$= 0.443$$

$$= Gini_{income \in \{high\}}(D).$$

$Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index

- All attributes are assumed continuous-valued
- May need other tools, e.g., clustering, to get the possible split values
- Can be modified for categorical attributes

Gini Index [CART] - Example

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- Lets now consider: **age:** {youth, middle_aged, senior}
  - Now consider each possible splitting subsets

{{youth, middle_aged}, {youth, senior}, {middle_aged, senior}, {youth},{middle_aged}, {senior}}

age	income	student	credit_rating	buys_computer
youth	high	no	fair	no
youth	high	no	excellent	no
middle_aged	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle_aged	low	yes	excellent	yes
youth	medium	no	fair	no
youth	low	yes	fair	yes
senior	medium	yes	fair	yes
youth	medium	yes	excellent	yes
middle_aged	medium	no	excellent	yes
middle_aged	high	yes	fair	yes
senior	medium	no	excellent	no

$$gini_{age \in \{youth, middle\_aged\}}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{9}{14}\left(1 - \left(\frac{6}{9}\right)^2 - \left(\frac{3}{9}\right)^2\right) + \frac{5}{14}\left(1 - \left(\frac{3}{5}\right)^2 - \left(\frac{2}{5}\right)^2\right) = \mathbf{0.4571}$$

$$= gini_{age \in \{senior\}}(D)$$

$$gini_{age \in \{youth, senior\}}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{5}{10}\right)^2 - \left(\frac{5}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{4}{4}\right)^2 - \left(\frac{0}{4}\right)^2\right) = \mathbf{0.3571}$$

$$= gini_{age \in \{middle\_aged\}}(D)$$

$$gini_{age \in \{middle\_aged, senior\}}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{9}{14}\left(1 - \left(\frac{7}{9}\right)^2 - \left(\frac{2}{9}\right)^2\right) + \frac{5}{14}\left(1 - \left(\frac{2}{5}\right)^2 - \left(\frac{3}{5}\right)^2\right) = \mathbf{0.3936} = gini_{age \in \{youth\}}(D)$$

		Class		
		yes	no	
age	youth	2	3	5
	middle_aged	4	0	4
	senior	3	2	5
				14

Gini Index [CART] - Example

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- Lets first consider: **income:** {low, medium, high}
  - Now consider each possible splitting subsets

{{low, medium}, {low, high}, {medium, high}, {low}, {medium}, {high}}

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) = \mathbf{0.4428}$$

$$= gini_{income \in \{high\}}(D)$$

$$gini_{income \in \{low,high\}}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{8}{14}\left(1 - \left(\frac{5}{8}\right)^2 - \left(\frac{3}{8}\right)^2\right) + \frac{6}{14}\left(1 - \left(\frac{4}{6}\right)^2 - \left(\frac{2}{6}\right)^2\right) = \mathbf{0.4583}$$

$$= gini_{income \in \{medium\}}(D)$$

$$gini_{income \in \{medium,high\}}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{10}{14}\left(1 - \left(\frac{6}{10}\right)^2 - \left(\frac{4}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{3}{4}\right)^2 - \left(\frac{1}{4}\right)^2\right) = \mathbf{0.45} = gini_{income \in \{low\}}(D)$$

age	income	student	credit_rating	buys_computer
youth	high	no	fair	no
youth	high	no	excellent	no
middle_aged	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle_aged	low	yes	excellent	yes
youth	medium	no	fair	no
youth	low	yes	fair	yes
senior	medium	yes	fair	yes
youth	medium	yes	excellent	yes
middle_aged	medium	no	excellent	yes
middle_aged	high	yes	fair	yes
senior	medium	no	excellent	no

		Class		
		yes	no	
income	low	3	1	4
	medium	4	2	6
	high	2	2	4
		9	5	14

Gini Index [CART] - Example

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

$$gini_A(D) = \frac{|D_1|}{|D|} gini(D_1) + \frac{|D_2|}{|D|} gini(D_2)$$

$$\Delta gini(A) = gini(D) - gini_A(D)$$

- Lets now consider: **student**
  - It is a binary attribute

		Class		
		yes	no	
student	yes	6	1	7
	no	3	4	7
				14

$$gini_{student}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{7}{14}\left(1 - \left(\frac{6}{7}\right)^2 - \left(\frac{1}{7}\right)^2\right) + \frac{7}{14}\left(1 - \left(\frac{3}{7}\right)^2 - \left(\frac{4}{7}\right)^2\right) = 0.3673$$

- Lets now consider: **credit_rating**
  - It is a binary attribute

$$gini_{credit-rating}(D) = \left(\frac{D_1}{14}\right) gini(D_1) + \left(\frac{D_2}{14}\right) gini(D_2)$$

$$= \frac{8}{14}\left(1 - \left(\frac{6}{8}\right)^2 - \left(\frac{2}{8}\right)^2\right) + \frac{6}{14}\left(1 - \left(\frac{3}{6}\right)^2 - \left(\frac{3}{6}\right)^2\right) = 0.4285$$

age	income	student	credit_rating	buys_computer
youth	high	no	fair	no
youth	high	no	excellent	no
middle_aged	high	no	fair	yes
senior	medium	no	fair	yes
senior	low	yes	fair	yes
senior	low	yes	excellent	no
middle_aged	low	yes	excellent	yes
youth	medium	no	fair	no
youth	low	yes	fair	yes
senior	medium	yes	fair	yes
youth	medium	yes	excellent	yes
middle_aged	medium	no	excellent	yes
middle_aged	high	yes	fair	yes
senior	medium	no	excellent	no

		Class		
		yes	no	
credit_rating	fair	6	2	8
	excellent	3	3	6
				14

Gini Index [CART] - Example

Attribute	Split	Gini index	Reduction in impurity $\Delta gini = gini(D) - gini_A(D)$
age	{youth, senior} or {middle_aged}	0.3571	0.459 - 0.3571 = 0.1019
income	{medium, high} or {low}	0.4428	0.459 - 0.4428 = 0.0162
student	Binary	0.3673	0.459 - 0.3673 = 0.0917
credit_rating	Binary	0.4285	0.459 - 0.4285 = 0.0305

Gini Index [CART] - Example

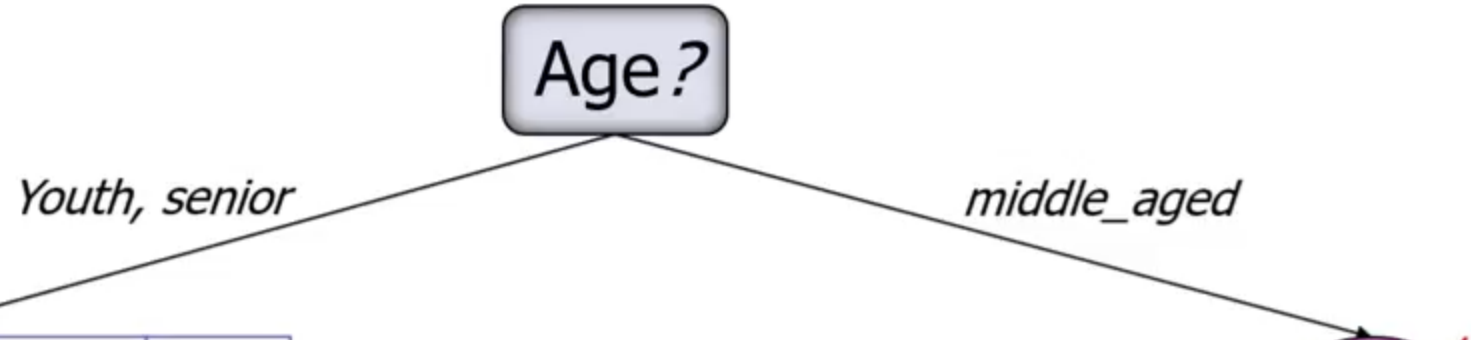
Youth, senior

income	student	credit_rating	class
high	no	Fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	Yes
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

middle_aged

income	student	credit_rating	class
high	no	fair	yes
low	yes	excellent	yes
medium	no	excellent	yes
high	yes	fair	yes

Gini Index [CART] - Example



income	student	credit_rating	class
high	no	Fair	no
high	no	excellent	no
medium	no	fair	no
low	yes	fair	yes
medium	yes	excellent	Yes
medium	no	fair	yes
low	yes	fair	yes
low	yes	excellent	no
medium	yes	fair	yes
medium	no	excellent	no

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

Overfitting and Tree Pruning

- Overfitting: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *Remove branches* from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

Enhancements to Basic Decision Tree Induction

- Allow for **continuous-valued attributes**
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle **missing attribute values**
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- **Attribute construction**
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

Classification in Large Databases

- Classification—a classical problem extensively studied by statisticians and machine learning researchers
- Scalability: Classifying data sets with millions of examples and hundreds of attributes with reasonable speed
- Why is decision tree induction popular?
  - relatively faster learning speed (than other classification methods)
  - convertible to simple and easy to understand classification rules
  - can use SQL queries for accessing databases
  - comparable classification accuracy with other methods
- RainForest (VLDB'98 — Gehrke, Ramakrishnan & Ganti)
  - Builds an AVC-list (attribute, value, class label)

Support Vector Machines (SVM)

UNIT II

SVM—Support Vector Machines

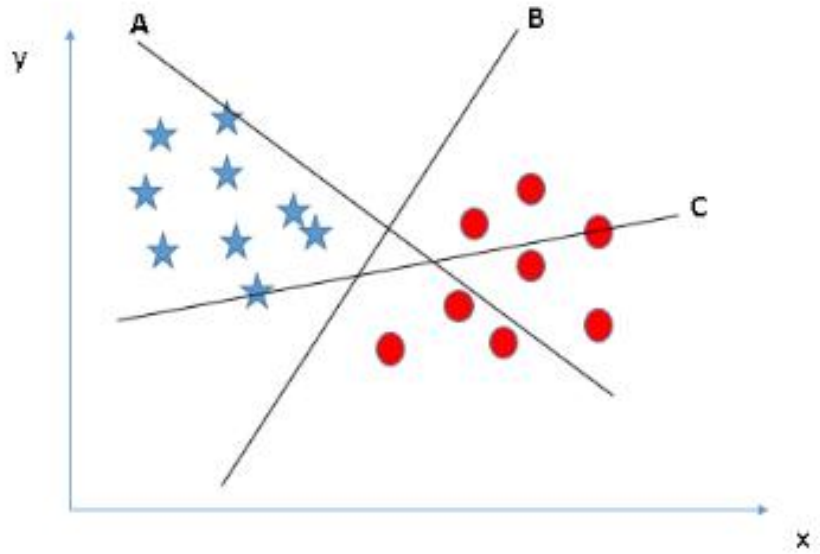
- A relatively new classification method for both linear and nonlinear data
- It uses a nonlinear mapping to transform the original training data into a higher dimension
- With the new dimension, it searches for the linear optimal separating **hyperplane** (i.e., “decision boundary”)
- With an appropriate nonlinear mapping to a sufficiently high dimension, data from two classes can always be separated by a hyperplane
- SVM finds this hyperplane using **support vectors** (“essential” training tuples) and **margins** (defined by the support vectors)

SVM

- “Support Vector Machine” (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges.
- it is mostly used in classification problems.
- In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate.
- Then, we perform classification by finding the hyper-plane that differentiates the two classes very well (look at the below snapshot).

How does it work?

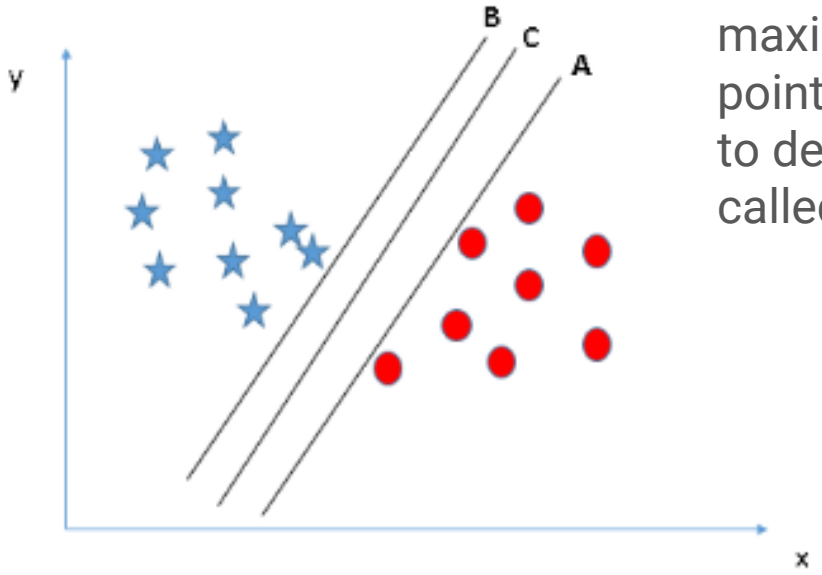
Identify the right hyper-plane (Scenario-1): Here, we have three hyper-planes (A, B and C). Now, identify the right hyper-plane to classify star and circle.



A thumb rule to identify the right hyper-plane: “Select the hyper-plane which segregates the two classes better”. In this scenario, hyper-plane “B” has excellently performed this job.

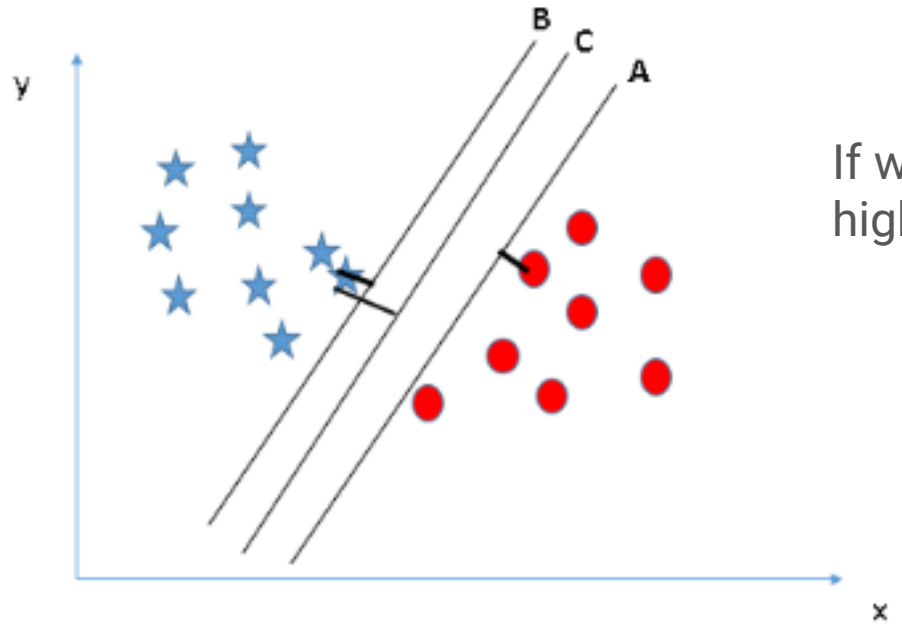
How does it work?

Identify the right hyper-plane (Scenario-2): Here, we have three hyper-planes (A, B and C) and all are segregating the classes well. Now, How can we identify the right hyper-plane?



maximizing the distances between nearest data point (either class) and hyper-plane will help us to decide the right hyper-plane. This distance is called as **Margin**.

Margin in SVM

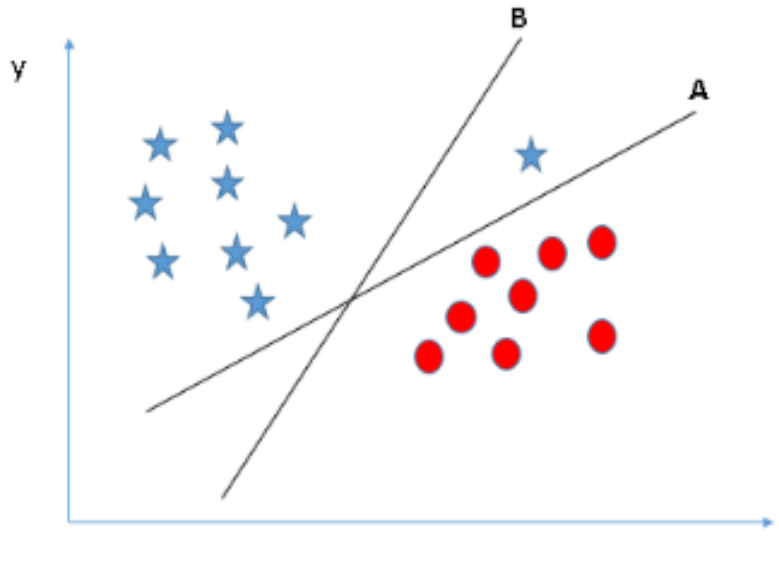


Margin for hyper-plane C is high as compared to both A and B. Hence, we name the right hyper-plane as C.

If we select a hyper-plane having low margin then there is high chance of miss-classification.

How does it Work?

Identify the right hyper-plane (Scenario-3): Hint: Use the rules as discussed in previous section to identify the right hyper-plane



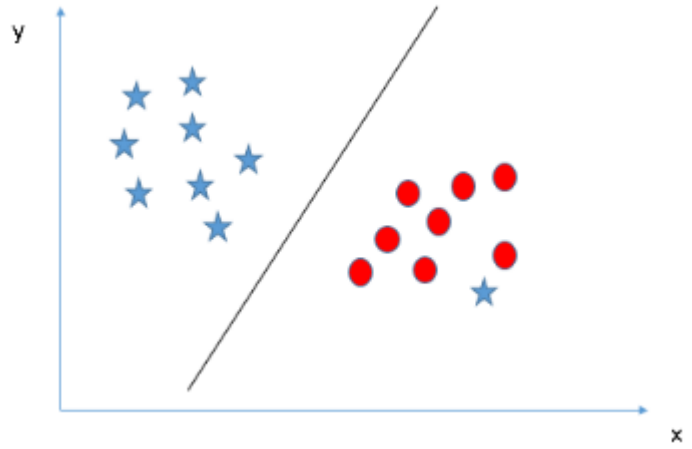
Hyper-plane **B** as it has higher margin compared to **A.** But, here is the catch, SVM selects the hyper-plane which classifies the classes accurately prior to maximizing margin.

Here, hyper-plane B has a classification error and A has classified all correctly. Therefore, the right hyper-plane is **A.**

How does it work?

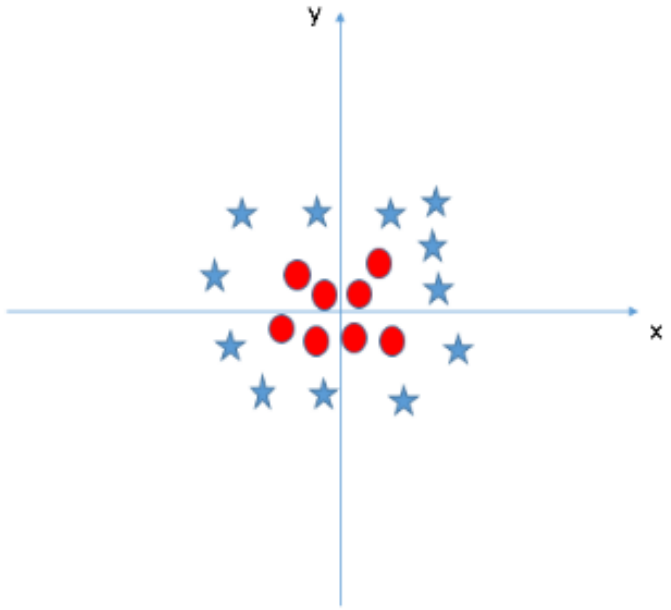
Can we classify two classes (Scenario-4)?: Below, I am unable to segregate the two classes using a straight line, as one of the stars lies in the territory of other(circle) class as an outlier.

The SVM algorithm has a feature to ignore outliers and find the hyper-plane that has the maximum margin.
Hence, we can say, SVM classification is robust to outliers.

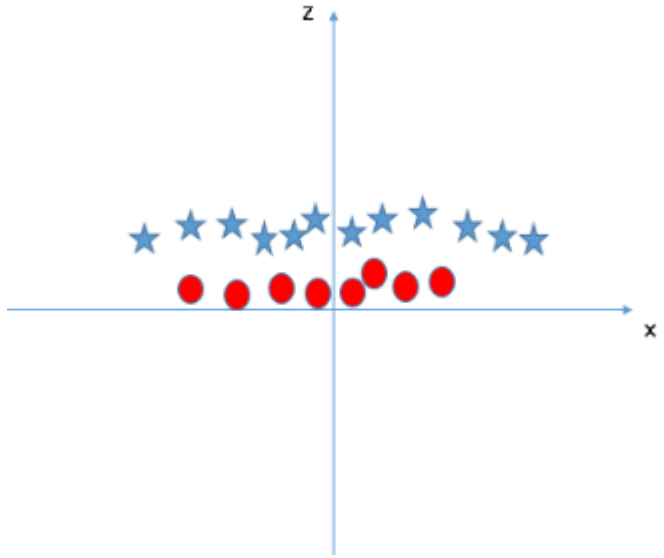


How does it work?

Find the hyper-plane to segregate to classes (Scenario-5): In the scenario below, we can't have linear hyper-plane between the two classes, so how does SVM classify these two classes? Till now, we have only looked at the linear hyper-plane.



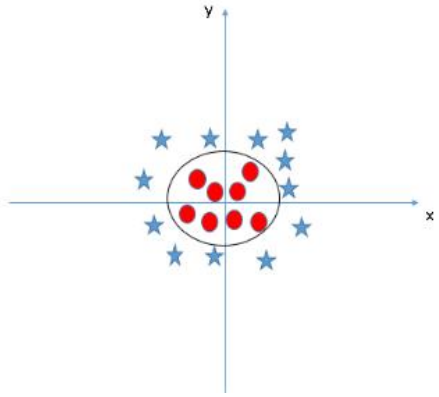
SVM can solve this problem. Easily! It solves this problem by introducing additional feature. Here, we will add a new feature z=x^2+y^2. Now, let's plot the data points on axis x and z:



kernel trick

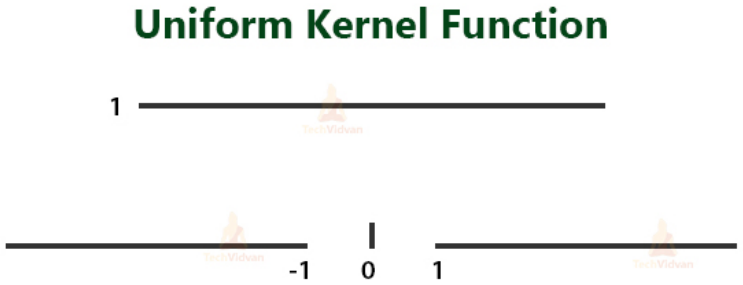
- The SVM kernel is a function that takes low dimensional input space and transforms it to a higher dimensional space i.e. it converts not separable problem to separable problem.
- it does some extremely complex data transformations, then finds out the process to separate the data based on the labels or outputs you've defined.

When we look at the hyper-plane in original input space it looks like a circle:

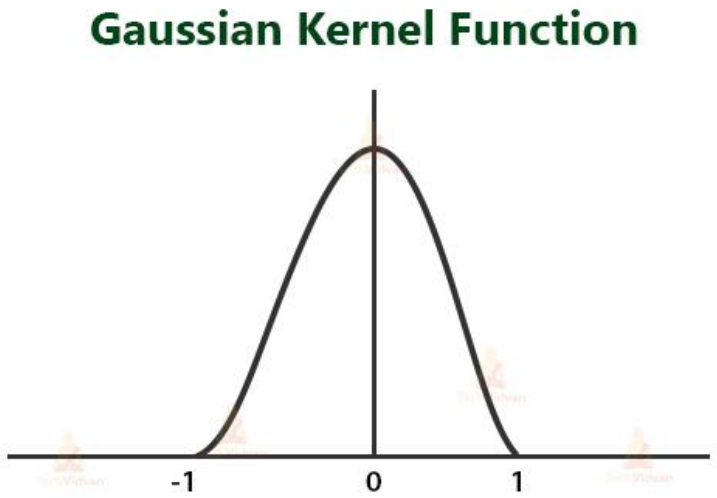


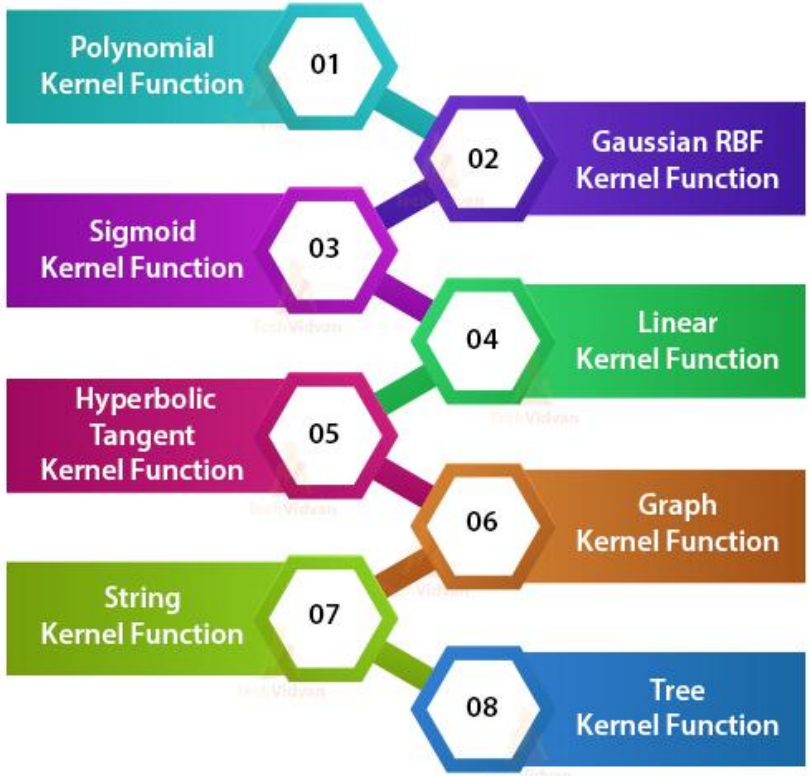
Different Kernel Functions

Uniform kernels: $K(x) = I(||x|| <= 1)$



Gaussian kernels: $K(x) = \exp(-||x||^2)$





Linear	$K(x, y) = x \cdot y$
Sigmoid	$K(x, y) = \tanh(ax.y + b)$
Polynomial	$K(x, y) = (1 + x \cdot y)^d$
KMOD	$K(x, y) = a\left[\exp\left(\frac{\gamma}{\|\|x-y\|\|^2+\sigma^2}\right) - 1\right]$
RBF	$K(x, y) = \exp(-a\|\|x - y\|\|^2)$
Exponential RBF	$K(x, y) = \exp(-a\|\|x - y\|\|)$

SVM—Linearly Separable

- A separating hyperplane can be written as

  $\mathbf{W} \bullet \mathbf{X} + b = 0$

  where $\mathbf{W}=\{w_1, w_2, \ldots, w_n\}$ is a weight vector and b a scalar (bias)
- For 2-D it can be written as

  $w_0 + w_1 x_1 + w_2 x_2 = 0$
- The hyperplane defining the sides of the margin:

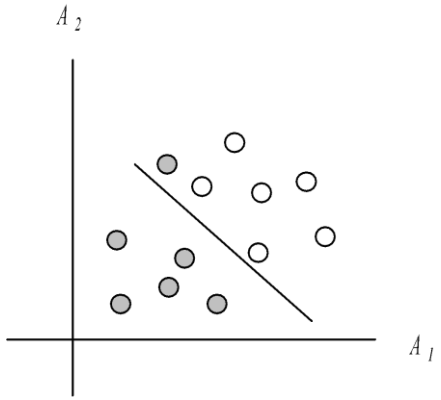
  $H_1: w_0 + w_1 x_1 + w_2 x_2 \geq 1$ for $y_i = +1$, and

  $H_2: w_0 + w_1 x_1 + w_2 x_2 \leq -1$ for $y_i = -1$
- Any training tuples that fall on hyperplanes H_1 or H_2 (i.e., the sides defining the margin) are **support vectors**
- This becomes a **constrained (convex) quadratic optimization** problem: Quadratic objective function and linear constraints ☐ *Quadratic Programming (QP)* ☐ Lagrangian multipliers

Why Is SVM Effective on High Dimensional Data?

- The **complexity** of trained classifier is characterized by the # of support vectors rather than the dimensionality of the data
- The **support vectors** are the essential or critical training examples — they lie closest to the decision boundary (MMH)
- If all other training examples are removed and the training is repeated, the same separating hyperplane would be found
- The number of support vectors found can be used to compute an (upper) bound on the expected error rate of the SVM classifier, which is independent of the data dimensionality
- Thus, an SVM with a small number of support vectors can have good generalization, even when the dimensionality of the data is high

SVM—Linearly Inseparable



- Transform the original input data into a higher dimensional space

Example 6.8 Nonlinear transformation of original input data into a higher dimensional space. Consider the following example. A 3D input vector $\mathbf{X} = (x_1, x_2, x_3)$ is mapped into a 6D space Z using the mappings $\phi_1(X) = x_1, \phi_2(X) = x_2, \phi_3(X) = x_3, \phi_4(X) = (x_1)^2, \phi_5(X) = x_1x_2$, and $\phi_6(X) = x_1x_3$. A decision hyperplane in the new space is $d(\mathbf{Z}) = \mathbf{WZ} + b$, where $\mathbf{W}$ and $\mathbf{Z}$ are vectors. This is linear. We solve for $\mathbf{W}$ and b and then substitute back so that we see that the linear decision hyperplane in the new $(\mathbf{Z})$ space corresponds to a nonlinear second order polynomial in the original 3-D input space,

$$\begin{aligned} d(Z) &= w_1x_1 + w_2x_2 + w_3x_3 + w_4(x_1)^2 + w_5x_1x_2 + w_6x_1x_3 + b \\ &= w_1z_1 + w_2z_2 + w_3z_3 + w_4z_4 + w_5z_5 + w_6z_6 + b \end{aligned}$$

■

- Search for a linear separating hyperplane in the new space

SVM: Different Kernel functions

- Instead of computing the dot product on the transformed data, it is math. equivalent to applying a kernel function $K(\mathbf{X_i}, \mathbf{X_j})$ to the original data, i.e., $K(\mathbf{X_i}, \mathbf{X_j}) = \Phi(\mathbf{X_i})\ \Phi(\mathbf{X_j})$
- Typical Kernel Functions

$$\textbf{Polynomial kernel of degree } h: \quad K(X_i, X_j) = (X_i \cdot X_j + 1)^h$$

$$\textbf{Gaussian radial basis function kernel}: \quad K(X_i, X_j) = e^{-\|X_i - X_j\|^2/2\sigma^2}$$

$$\textbf{Sigmoid kernel}: \quad K(X_i, X_j) = \tanh(\kappa X_i \cdot X_j - \delta)$$

- SVM can also be used for classifying multiple (> 2) classes and for regression analysis (with additional parameters)

Pros and Cons associated with SVM

- **Pros:**
  - It works really well with a clear margin of separation
  - It is effective in high dimensional spaces.
  - It is effective in cases where the number of dimensions is greater than the number of samples.
  - It uses a subset of training points in the decision function (called support vectors), so it is also memory efficient.

- **Cons:**
  - It doesn't perform well when we have large data set because the required training time is higher
  - It also doesn't perform very well, when the data set has more noise i.e. target classes are overlapping
  - SVM doesn't directly provide probability estimates, these are calculated using an expensive five-fold cross-validation. It is included in the related SVC method of Python scikit-learn library.

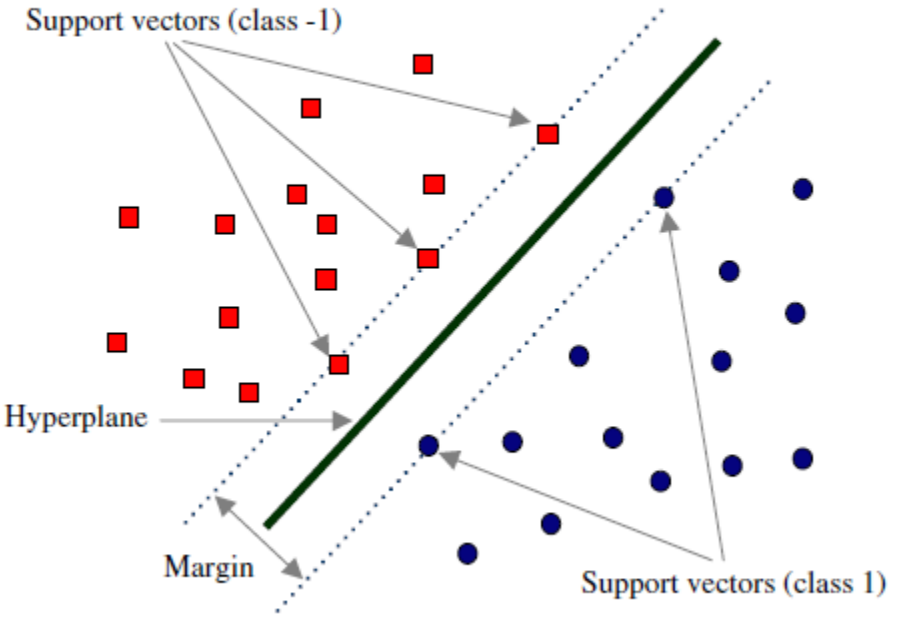
SVM in Python

```
# import support vector classifier
from sklearn.svm import SVC # "Support Vector Classifier"
clf = SVC(kernel='linear')

# fitting x samples and y classes
clf.fit(x, y)
```

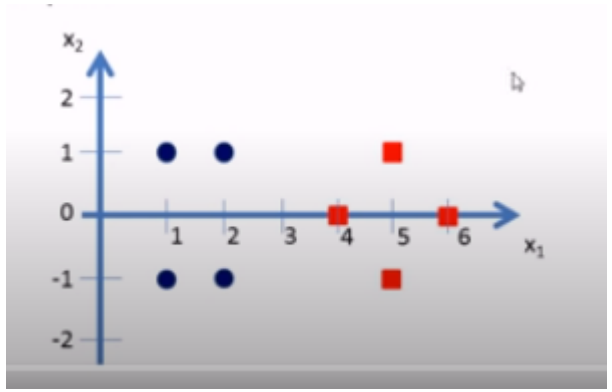
```
clf.predict([[120, 990]])

clf.predict([[85, 550]])
```

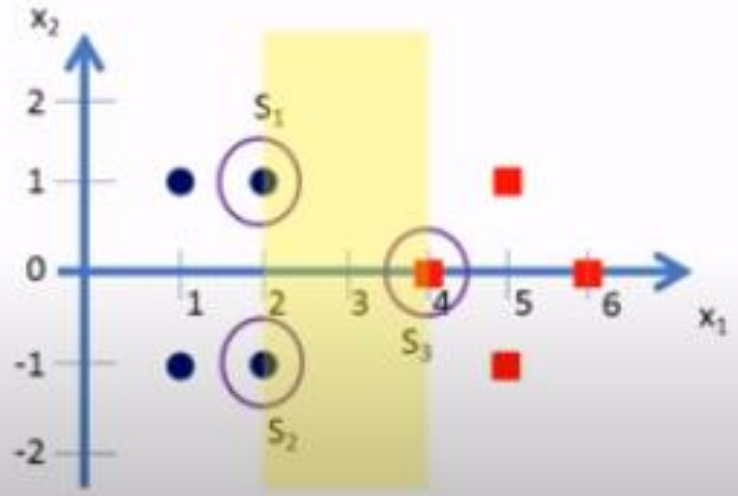


Exercise

- https://www.youtube.com/watch?v=LXGaYVXkGtg&t=189s



- Here we select 3 Support Vectors to start with.
- They are S_1, S_2 and S_3.



$$S_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix}$$

$$S_2 = \begin{pmatrix} 2 \\ -1 \end{pmatrix}$$

$$S_3 = \begin{pmatrix} 4 \\ 0 \end{pmatrix}$$

SVM

- Here we will use vectors augmented with a 1 as a bias input, and for clarity we will differentiate these with an over-tilde. That is:

$$s_1 = \begin{pmatrix} 2 \\ 1 \end{pmatrix} \qquad \tilde{s}_1 = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$$

$$s_2 = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \qquad \tilde{s}_2 = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$$

$$s_3 = \begin{pmatrix} 4 \\ 0 \end{pmatrix} \qquad \tilde{s}_3 = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$$

SVM

Now we need to find 3 parameters α_1, α_2, and α_3 based on the following 3 linear equations:

$$\alpha_1 \widetilde{S_1}.\widetilde{S_1} + \alpha_2 \widetilde{S_2}.\widetilde{S_1} + \alpha_3 \widetilde{S_3}.\widetilde{S_1} = -1 \ (-ve\ class)$$

$$\alpha_1 \widetilde{S_1}.\widetilde{S_2} + \alpha_2 \widetilde{S_2}.\widetilde{S_2} + \alpha_3 \widetilde{S_3}.\widetilde{S_2} = -1 \ (-ve\ class)$$

$$\alpha_1 \widetilde{S_1}.\widetilde{S_3} + \alpha_2 \widetilde{S_2}.\widetilde{S_3} + \alpha_3 \widetilde{S_3}.\widetilde{S_3} = +1 \ (+ve\ class)$$

Let's substitute the values for $\widetilde{S_1}$, $\widetilde{S_2}$ and $\widetilde{S_3}$ in the above equations.

$$\widetilde{S_1} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} \quad \widetilde{S_2} = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} \quad \widetilde{S_3} = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$$

$$\alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}.\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}.\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} = -1$$

$$\alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}.\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}.\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} = -1$$

$$\alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}.\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}.\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}.\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} = +1$$

After simplification we get:

$$6\alpha_1 + 4\alpha_2 + 9\alpha_3 = -1$$

$$4\alpha_1 + 6\alpha_2 + 9\alpha_3 = -1$$

$$9\alpha_1 + 9\alpha_2 + 17\alpha_3 = +1$$

Scholastic
home.video.tutor
@gmail.com

$\alpha_1=\alpha_2=$ -3.25 and $\alpha_3=$ 3.5

$\widetilde{S_1} = \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix}$

$\widetilde{S_2} = \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$

$\widetilde{S_3} = \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$

- The hyper plane that discriminates the positive class from the negative class is give by:

$$\widetilde{w} = \sum_i \alpha_i \widetilde{S_i}$$

- Substituting the values we get:

$$\widetilde{w} = \alpha_1 \begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + \alpha_2 \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + \alpha_3 \begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix}$$

$$\widetilde{w} = (-3.25).\begin{pmatrix} 2 \\ 1 \\ 1 \end{pmatrix} + (-3.25).\begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix} + (3.5).\begin{pmatrix} 4 \\ 0 \\ 1 \end{pmatrix} = \begin{pmatrix} 1 \\ 0 \\ -3 \end{pmatrix}$$

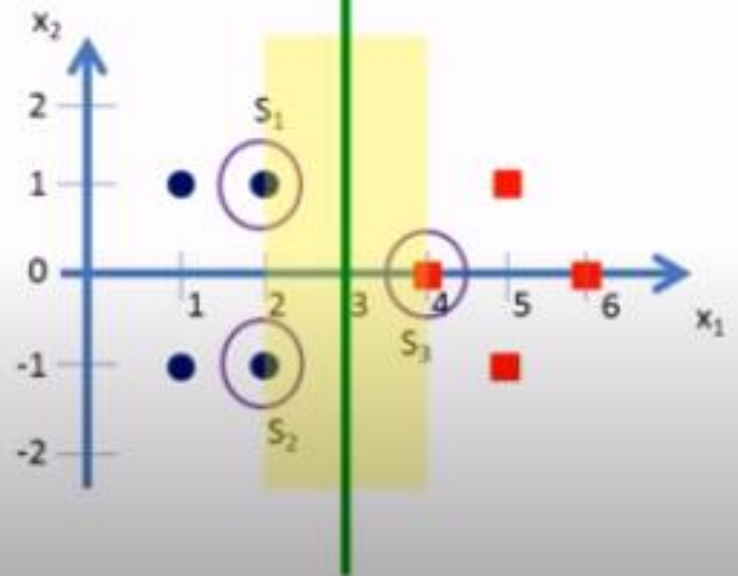
Our vectors are augmented with a bias.

Hence we can equate the entry in $\widetilde{w}$ as the hyper plane with an offset b.

Therefore the separating hyper plane equation

$y = wx + b$ with $w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and offset $b = -3$.

$y = wx + b$ with $w = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$ and offset $b = -3$.



Bayesian Classification

- A statistical classifier: performs *probabilistic prediction, i.e.,* predicts class membership probabilities
- Foundation: Based on Bayes' Theorem.
- Performance: A simple Bayesian classifier, *naïve Bayesian classifier*, has comparable performance with decision tree and selected neural network classifiers
- Incremental: Each training example can incrementally increase/decrease the probability that a hypothesis is correct — prior knowledge can be combined with observed data
- Standard: Even when Bayesian methods are computationally intractable, they can provide a standard of optimal decision making against which other methods can be measured

Bayesian Classification - Example

age	income	student	credit_rating	com
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Bayesian Classification - Probability

- *Probability:* How likely something is to happen

- Probability of an event happening =

$$\frac{\textit{Number of times it can happen}}{\textit{Total number of outcomes}}$$

Bayesian Classification Theorem

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(H|**X**), the probability that the hypothesis holds given the observed data sample **X**
- P(H) (*prior probability*), the initial probability
  - E.g., **X** will buy computer, regardless of age, income, …
- P(**X**): probability that sample data is observed
- P(**X**|H) (*posteriori probability*), the probability of observing the sample **X**, given that the hypothesis holds
  - E.g., Given that **X** will buy computer, the prob. that X is 31..40, medium income

Bayesian Classification Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes theorem
-

$$P(H \mid \mathbf{X}) = \frac{P(\mathbf{X} \mid H)P(H)}{P(\mathbf{X})}$$

- ***Predicts X belongs to C_i iff the probability $P(C_i|X)$ is the highest among all the $P(C_k|X)$ for all the k classes***
- Practical difficulty: require initial knowledge of many probabilities, significant computational cost

Naïve Bayesian Classification Theorem

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = (x_1, x_2, …, x_n)
- Suppose there are *m* classes C_1, C_2, …, C_m.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- This can be derived from Bayes' theorem
- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Naïve Bayesian Classification - Example

Class:
C1:buys_computer = 'yes'
C2:buys_computer = 'no'

Data sample
X = (age = youth,
income = medium,
student = yes
credit_rating = fair)

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

Naïve Bayesian Classification - Example

Test for X = (age = youth , income = medium, student = yes, credit_rating = fair)

- Prior probability $P(C_i)$: **P(buys_computer = yes) = 9/14 = 0.643** **P(buys_computer = no) = 5/14= 0.357**
- To compute $P(X|C_i)$ for each class, compute following conditional probabilities

P(age = youth| buys_computer = yes) = 2/9 = 0.222

P(age = youth | buys_computer = no) = 3/5 = 0.6

P(income = medium | buys_computer = yes) = 4/9 = 0.444

P(income = medium | buys_computer = no) = 2/5 = 0.4

P(student = yes | buys_computer = yes) = 6/9 = 0.667

P(student = yes | buys_computer = no) = 1/5 = 0.2

P(credit_rating = fair | buys_computer = yes) = 6/9 = 0.667

P(credit_rating = fair | buys_computer = no) = 2/5 = 0.4

RID	age	income	student	credit_rating	Class: buys_computer
1	youth	high	no	fair	no
2	youth	high	no	excellent	no
3	middle_aged	high	no	fair	yes
4	senior	medium	no	fair	yes
5	senior	low	yes	fair	yes
6	senior	low	yes	excellent	no
7	middle_aged	low	yes	excellent	yes
8	youth	medium	no	fair	no
9	youth	low	yes	fair	yes
10	senior	medium	yes	fair	yes
11	youth	medium	yes	excellent	yes
12	middle_aged	medium	no	excellent	yes
13	middle_aged	high	yes	fair	yes
14	senior	medium	no	excellent	no

- **$P(X|C_i)$:** P(X|buys_computer = yes) = 0.222 x 0.444 x 0.667 x 0.667 = 0.044

  P(X|buys_computer = no) = 0.6 x 0.4 x 0.2 x 0.4 = 0.019

- **$P(X|C_i)*P(C_i)$:** P(X|buys_computer = yes) * P(buys_computer = yes) = **0.028**

  P(X|buys_computer = no) * P(buys_computer = no) = **0.007**

0.028>0.007 ..

Therefore, X belongs to class ("buys_computer = yes")

Comment on Naive Bayes Classification

- Advantages
  - Easy to implement
  - Good results obtained in most of the cases
- Disadvantages
  - Assumption: class conditional independence i.e. effect of an attribute value on a given class is independent of the values of other attributes, therefore loss of accuracy

Lazy vs. Eager Learning

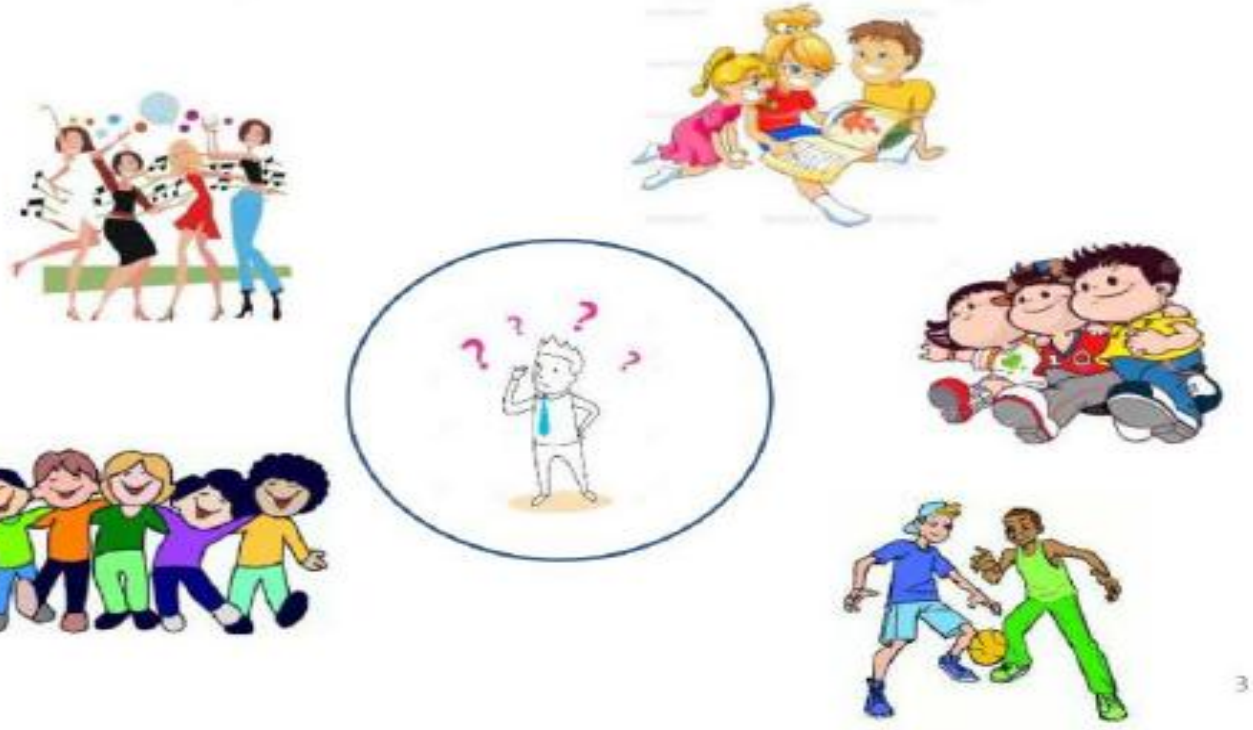
- Lazy vs. eager learning
  - **Lazy learning** (e.g., instance-based learning): Simply stores training data (or only minor processing) and waits until it is given a test tuple
  - **Eager learning** (the above discussed methods): Given a set of training tuples, constructs a classification model before receiving new (e.g., test) data to classify
- Lazy: less time in training but more time in predicting
- Accuracy
  - Lazy method effectively uses a richer hypothesis space since it uses many local linear functions to form an implicit global approximation to the target function
  - Eager: must commit to a single hypothesis that covers the entire instance space

Lazy Learner: Instance-Based Methods

- Instance-based learning:
  - Store training examples and delay the processing ("lazy evaluation") until a new instance must be classified
- Typical approaches
  - *k*-nearest neighbor approach
    - Instances represented as points in a Euclidean space.
  - Locally weighted regression
    - Constructs local approximation
  - Case-based reasoning
    - Uses symbolic representations and knowledge-based inference

Simple Analogy..

- Tell me about your friends(*who your neighbors are*) and *I will tell you who you are*.



Instance-based Learning

KNN – Different names

- K-Nearest Neighbors
- Memory-Based Reasoning
- Example-Based Reasoning
- Instance-Based Learning
- Lazy Learning

K-NN Classification

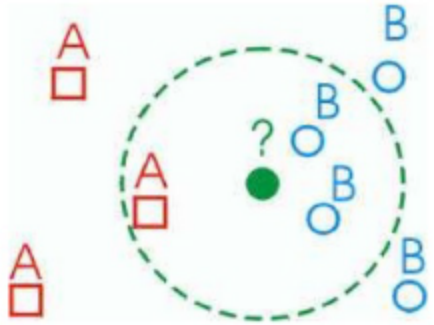
What is KNN?

- A powerful classification algorithm used in pattern recognition.
- K nearest neighbors stores all available cases and classifies new cases based on a *similarity measure*(e.g **distance function**)
- One of the top data mining algorithms used today.
- A non-parametric lazy learning algorithm (An Instance-based Learning method).

K-NN Classification

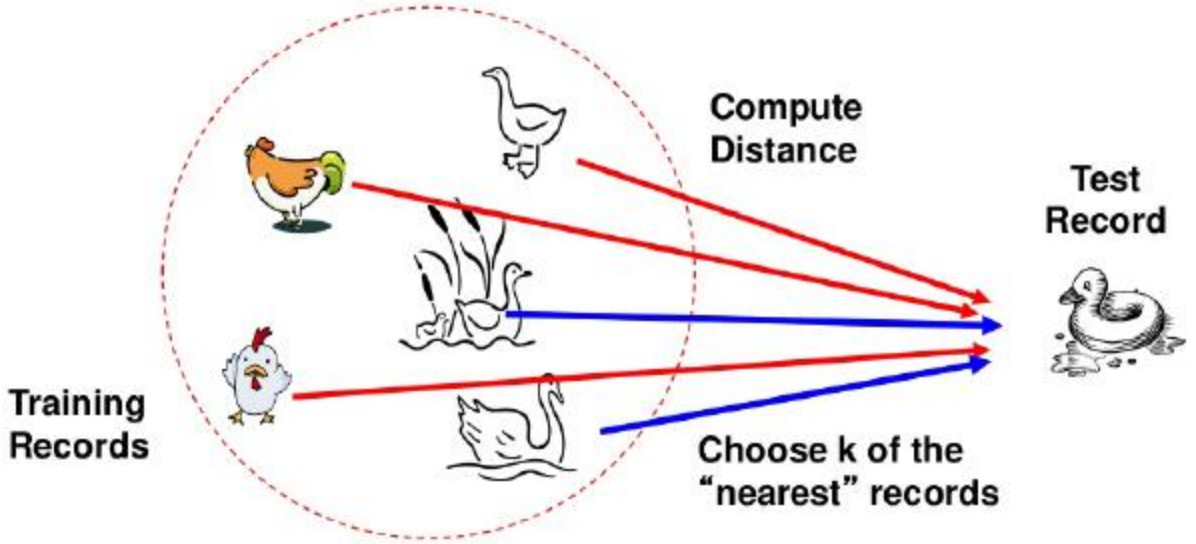
KNN: Classification Approach

- An object (a new instance) is classified by a majority votes for its neighbor classes.
- The object is assigned to the most common class amongst its K nearest neighbors.(*measured by a distant function*)



K-NN Classification

Distance Measure



K-NN Classification

Distance functions

Euclidean $\sqrt{\sum_{i=1}^{k}(x_i - y_i)^2}$

Manhattan $\sum_{i=1}^{k}|x_i - y_i|$

Minkowski $\left(\sum_{i=1}^{k}(|x_i - y_i|)^q\right)^{1/q}$

Euclidean distance is calculated as the square root of the sum of the squared differences between a new point (x) and an existing point (xi) across all input attributes j

Manhattan Distance: Calculate the distance between real vectors using the sum of their absolute difference

Minkowski Distance: Generalization of Euclidean and Manhattan distance

Euclidean is a good distance measure to use if the input variables are similar in type (e.g. all measured widths and heights). Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.).

K-NN Classification

Distance Between Neighbors

- Calculate the distance between new example (E) and all examples in the training set.

- *Euclidean* distance between two examples.
  - $X = [x_1, x_2, x_3, .., x_n]$
  - $Y = [y_1, y_2, y_3, \ldots, y_n]$
  - The Euclidean distance between *X* and *Y* is defined as:

$$D(X,Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

K-NN Classification

K-Nearest Neighbor Algorithm

- All the instances correspond to points in an n-dimensional feature space.
- Each instance is represented with a set of numerical attributes.
- Each of the training data consists of a set of vectors and a class label associated with each vector.
- Classification is done by comparing feature vectors of different K nearest points.
- Select the K-nearest examples to E in the training set.
- Assign E to the most common class among its K-nearest neighbors.

K-NN Classification

3-KNN: Example(1)

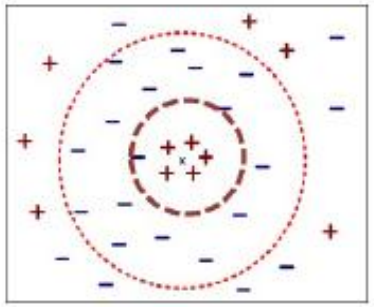
Customer	Age	Income	No. credit cards	Class
George	35	35K	3	No
Rachel	22	50K	2	Yes
Steve	63	200K	1	No
Tom	59	170K	1	No
Anne	25	40K	4	Yes
John	37	50K	2	YES

Distance from John
sqrt $[(35-37)^2+(35-50)^2+(3-2)^2]=15.16$
sqrt $[(22-37)^2+(50-50)^2+(2-2)^2]=15$
sqrt $[(63-37)^2+(200-50)^2+(1-2)^2]=152.23$
sqrt $[(59-37)^2+(170-50)^2+(1-2)^2]=122$
sqrt $[(25-37)^2+(40-50)^2+(4-2)^2]=15.74$

K-NN Classification

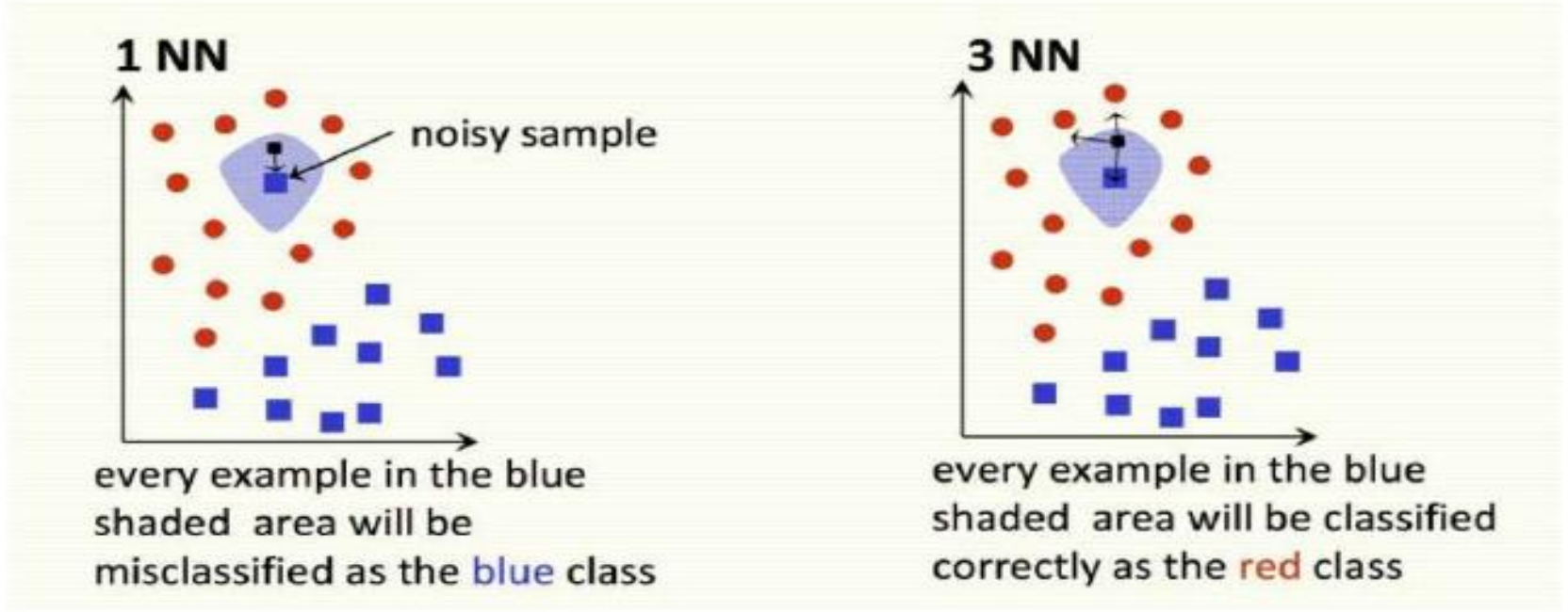
How to choose K?

- If K is too small it is sensitive to noise points.
- Larger K works well. But too large K may include majority points from other classes.

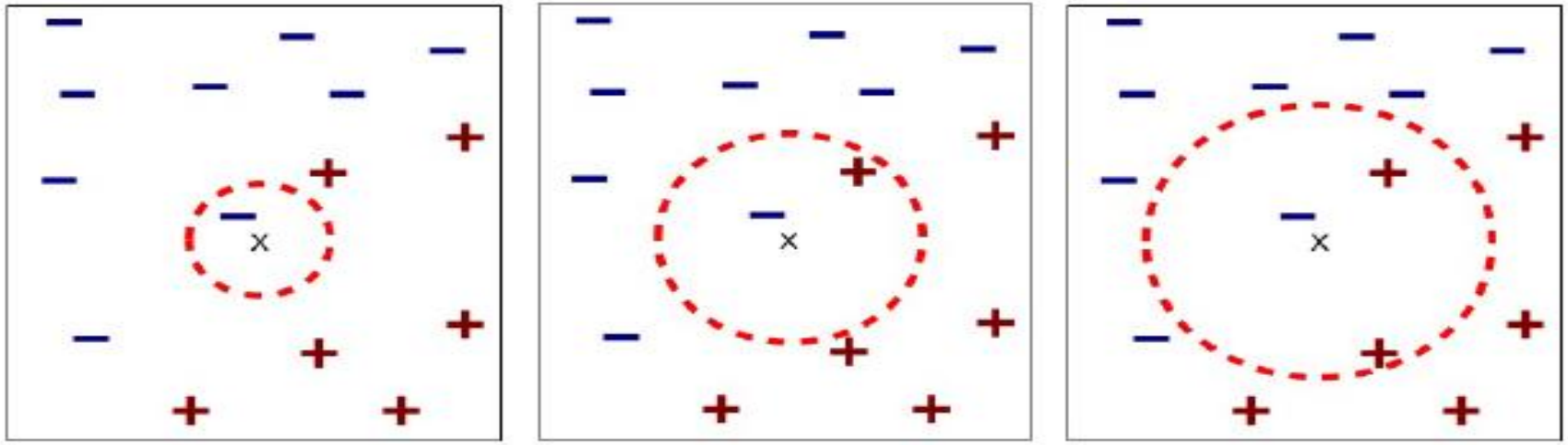


- Rule of thumb is K < sqrt(n), n is number of examples.

K-NN Classification



K-NN Classification



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

KNN Feature Weighting

- Scale each feature by its importance for classification

$$D(a,b) = \sqrt{\sum_k w_k (a_k - b_k)^2}$$

- Can use our prior knowledge about which features are more important
- Can learn the weights $\mathbf{w_k}$ using **cross-validation** (to be covered later)

Feature Normalization

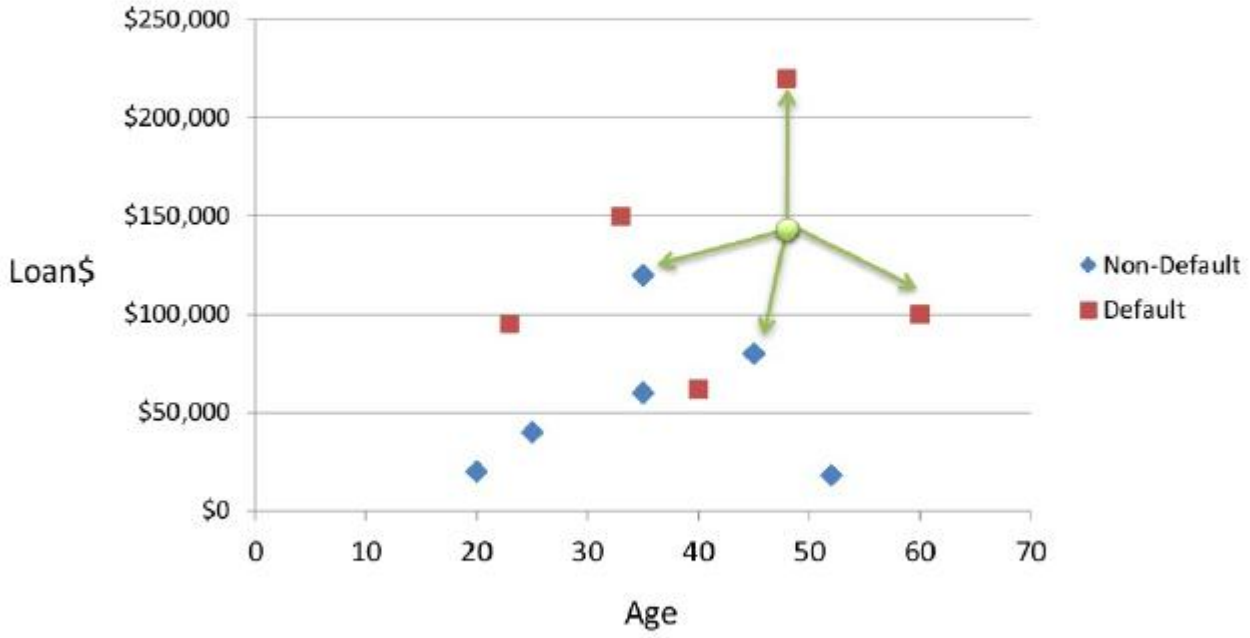
- Distance between neighbors could be dominated by some attributes with relatively large numbers.
  - e.g., income of customers in our previous example.

$$a_i = \frac{v_i - \min v_i}{\max v_i - \min v_i}$$

- Arises when two features are in different scales.

- Important to normalize those features.
  - Mapping values to numbers between 0 – 1.

K-NN Classification



K-NN Classification

Age	Loan	Default	Distance
25	$40,000	N	102000
35	$60,000	N	82000
45	$80,000	N	62000
20	$20,000	N	122000
35	$120,000	N	22000
52	$18,000	N	124000
23	$95,000	Y	47000
40	$62,000	Y	80000
60	$100,000	Y	42000
48	$220,000	Y	78000
33	$150,000	Y	8000
48	**$142,000**	**?**	

Euclidean Distance

$$D = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

K-NN Classification

Age	Loan	Default	Distance
0.125	0.11	N	0.7652
0.375	0.21	N	0.5200
0.625	0.31	N	0.3160
0	0.01	N	0.9245
0.375	0.50	N	0.3428
0.8	0.00	N	0.6220
0.075	0.38	Y	0.6669
0.5	0.22	Y	0.4437
1	0.41	Y	0.3650
0.7	1.00	Y	0.3861
0.325	0.65	Y	0.3771
0.7	0.61	?	

Standardized Variable

$$X_s = \frac{X - Min}{Max - Min}$$

K-NN Classification

Strengths of KNN

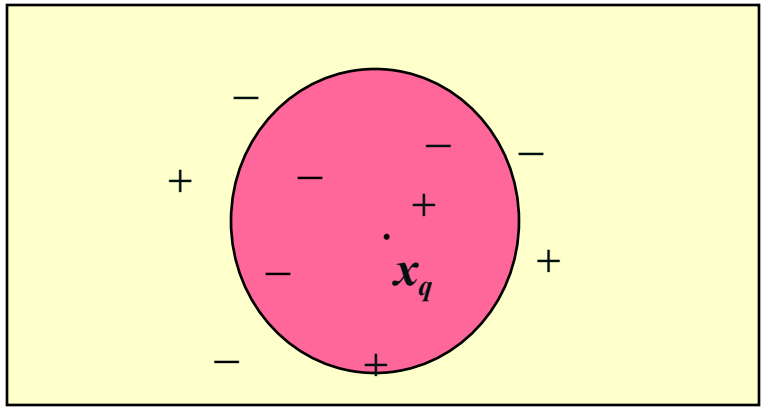
- Very simple and intuitive.
- Can be applied to the data from any distribution.
- Good classification if the number of samples is large enough.

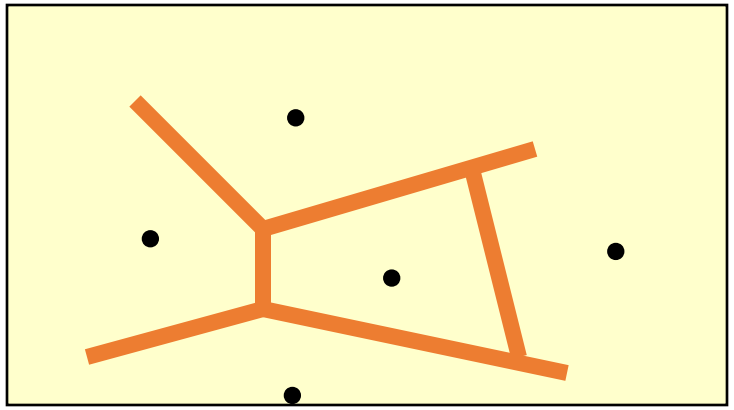
Weaknesses of KNN

- Takes more time to classify a new example.
  - need to calculate and compare distance from new example to all other examples.
- Choosing k may be tricky.
- Need large number of samples for accuracy.

The *k*-Nearest Neighbor Algorithm

- All instances correspond to points in the n-D space
- The nearest neighbor are defined in terms of Euclidean distance, dist($\mathbf{X}_1$, $\mathbf{X}_2$)
- Target function could be discrete- or real- valued
- For discrete-valued, *k*-NN returns the most common value among the *k* training examples nearest to x_q
- Vonoroi diagram: the decision surface induced by 1-NN for a typical set of training examples





Discussion on the *k*-NN Algorithm

- *k*-NN for real-valued prediction for a given unknown tuple
  - Returns the mean values of the *k* nearest neighbors
- Distance-weighted nearest neighbor algorithm
  - Weight the contribution of each of the *k* neighbors according to their distance to the query x_q
    - Give greater weight to closer neighbors

$$w \equiv \frac{1}{d(x_q, x_i)^2}$$

- Robust to noisy data by averaging *k*-nearest neighbors
- Curse of dimensionality: distance between neighbors could be dominated by irrelevant attributes
  - To overcome it, axes stretch or elimination of the least relevant attributes

https://www.kaggle.com/code/skalskip/iris-data-visualization-and-knn-classification

Feature Scaling

- **Feature scaling** is the method to limit the range of variables so that they can be compared on common grounds.

```
In [3]: dataset
```

Out[3]:

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

Methods [edit]

Rescaling (min-max normalization) [edit]

Also known as min-max scaling or min-max normalisation, is the simplest method and consists in rescaling the range of features to scale the range in [0, 1] or [−1, 1]. Selecting the target range depends on the nature of the data. The general formula is given as:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

where x is an original value, x' is the normalized value. For example, suppose that we have the students' weight data, and the students' weights span [160 pounds, 200 pounds]. To rescale this data, we first subtract 160 from each student's weight and divide the result by 40 (the difference between the maximum and minimum weights).

Mean normalization [edit]

$$x' = \frac{x - \text{average}(x)}{\max(x) - \min(x)}$$

where x is an original value, x' is the normalized value.

Standardization [edit]

In machine learning, we can handle various types of data, e.g. audio signals and pixel values for image data, and this data can include multiple dimensions. Feature standardization makes the values of each feature in the data have zero-mean (when subtracting the mean in the numerator) and unit-variance. This method is widely used for normalization in many machine learning algorithms (e.g., support vector machines, logistic regression, and artificial neural networks)[2][*citation needed*]. The general method of calculation is to determine the distribution mean and standard deviation for each feature. Next we subtract the mean from each feature. Then we divide the values (mean is already subtracted) of each feature by its standard deviation.

$$x' = \frac{x - \bar{x}}{\sigma}$$

Where x is the original feature vector, $\bar{x}$ is the mean of that feature vector, and σ is its standard deviation.

In [23]:
```
from sklearn.preprocessing import StandardScaler
standard_X = StandardScaler()
```

In [24]:
```
X_train = standard_X.fit_transform(X_train)
X_test = standard_X.fit_transform(X_test)
```

dataset

Filter

	OCCUPATION	GENDER	AGE	EDUCATION_YEARS	WEEKLY_WORKING_HOURS	COMPANY_SIZE	SALARY
1	1.256289	0.621059	0.0000000	0.4580879	2.814231e-01	2.8419998	-1.1618950
2	-1.027872	0.621059	0.8841519	0.4580879	-1.924571e+00	-0.3665959	0.7745967
3	1.256289	0.621059	1.1123201	0.7634799	6.899404e-01	-0.2782626	-1.1618950
4	0.114208	0.621059	-0.4848575	-0.7634799	2.814231e-01	-0.3197875	0.7745967
5	0.114208	0.621059	1.2264042	-1.3742638	3.483229e-15	-0.3689812	0.7745967
6	1.256289	-1.449138	-1.2834462	0.7634799	1.098458e+00	-0.3693916	0.7745967
7	-1.027872	0.621059	-1.6256986	0.4580879	6.899404e-01	-0.2017020	-1.1618950
8	-1.027872	-1.449138	-0.5989416	0.7634799	2.814231e-01	-0.3557466	0.7745967
9	0.114208	-1.449138	0.7700677	-1.9850477	-1.679460e+00	-0.2942160	0.7745967
10	-1.027872	0.621059	0.0000000	0.4580879	2.814231e-01	-0.2873165	-1.1618950

Showing 1 to 10 of 10 entries

Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap
- Comparing classifiers:
  - Confidence intervals
  - Cost-benefit analysis and ROC Curves

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 boostrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 – 1/d)^d \approx e^{-1} = 0.368$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

Estimating Confidence Intervals: Classifier Models M_1 vs. M_2

- Suppose we have 2 classifiers, M_1 and M_2, which one is better?
- Use 10-fold cross-validation to obtain $\overline{err}(M_1)$ and $\overline{err}(M_2)$
- These mean error rates are just *estimates* of error on the true population of *future* data cases
- What if the difference between the 2 error rates is just attributed to *chance*?
  - Use a **test of statistical significance**
  - Obtain **confidence limits** for our error estimates

Estimating Confidence Intervals: Null Hypothesis

- Perform 10-fold cross-validation
- Assume samples follow a **t distribution** with *k–1* **degrees of freedom** (here, *k=10*)
- Use **t-test** (or **Student's t-test**)
- **Null Hypothesis**: M_1 & M_2 are the same
- If we can **reject** null hypothesis, then
  - we conclude that the difference between M_1 & M_2 is **statistically significant**
  - Chose model with lower error rate

Estimating Confidence Intervals: t-test

- If only 1 test set available: **pairwise comparison**
  - For i[th] round of 10-fold cross-validation, the same cross partitioning is used to obtain $err(M_1)_i$ and $err(M_2)_i$
  - Average over 10 rounds to get $\overline{err}(M_1)$ and $\overline{err}(M_2)$
  - **t-test** computes **t-statistic** with *k-1* **degrees of freedom:**

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/k}}$$ where

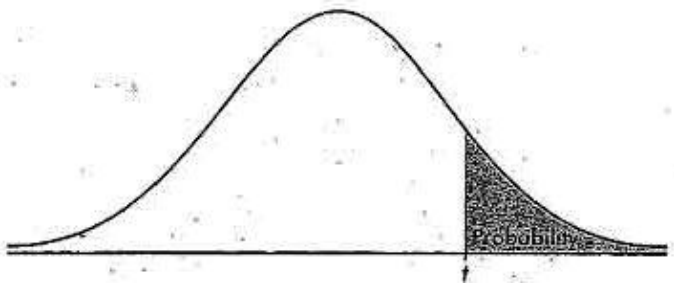
$$var(M_1 - M_2) = \frac{1}{k}\sum_{i=1}^{k}\left[err(M_1)_i - err(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))\right]^2$$

- If two test sets available: use **non-paired t-test**

where $$var(M_1 - M_2) = \sqrt{\frac{var(M_1)}{k_1} + \frac{var(M_2)}{k_2}},$$

where k_1 & k_2 are # of cross-validation samples used for M_1 & M_2, resp.

Estimating Confidence Intervals: Table for t-distribution



- Symmetric
- **Significance level**, e.g., *sig = 0.05* or *5%* means M_1 & M_2 are *significantly different* for 95% of population
- **Confidence limit**, *z = sig/2*

TABLE B: *t*-DISTRIBUTION CRITICAL VALUES

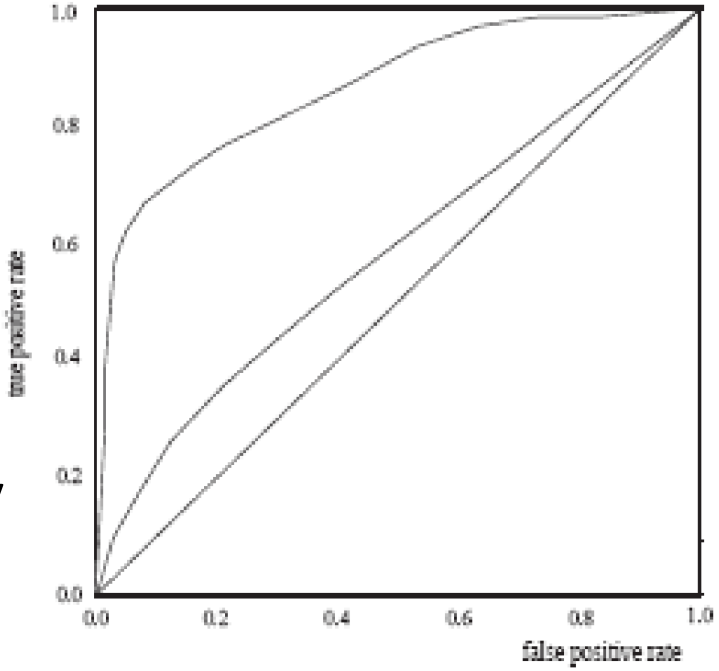
	Tail probability *p*											
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level *C*											

Estimating Confidence Intervals: Statistical Significance

- Are M_1 & M_2 **significantly different**?
  - Compute *t.* Select *significance level* (e.g. *sig = 5%)*
  - Consult table for t-distribution: Find *t value* corresponding to *k-1 degrees of freedom* (here, 9)
  - t-distribution is symmetric: typically upper % points of distribution shown → look up value for **confidence limit** *z=sig/2* (here, 0.025)
  - **If t > z or t < -z**, then t value lies in rejection region:
    - **Reject null hypothesis** that mean error rates of M_1 & M_2 are same
    - Conclude: statistically significant difference between M_1 & M_2
  - **Otherwise**, conclude that any difference is **chance**

Model Selection: ROC Curves

- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- Rank the test tuples in decreasing order: the one that is most likely to belong to the positive class appears at the top of the list
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model



- Vertical axis represents the true positive rate
- Horizontal axis rep. the false positive rate
- The plot also shows a diagonal line
- A model with perfect accuracy will have an area of 1.0

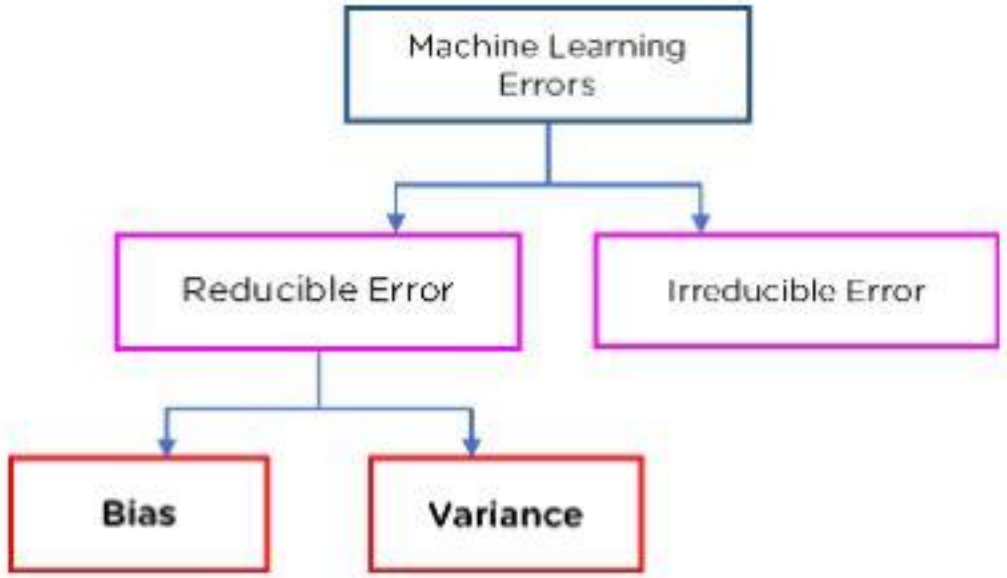
Issues Affecting Model Selection

- **Accuracy**
  - classifier accuracy: predicting class label
- **Speed**
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- **Robustness**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

Bias and variance

Errors in Machine Learning

- **Irreducible errors** are errors which will always be present in a machine learning model, because of unknown variables, and whose values cannot be reduced.
- **Reducible errors** are those errors whose values can be further reduced to improve a model. They are caused because our model's output function does not match the desired output function and can be optimized.

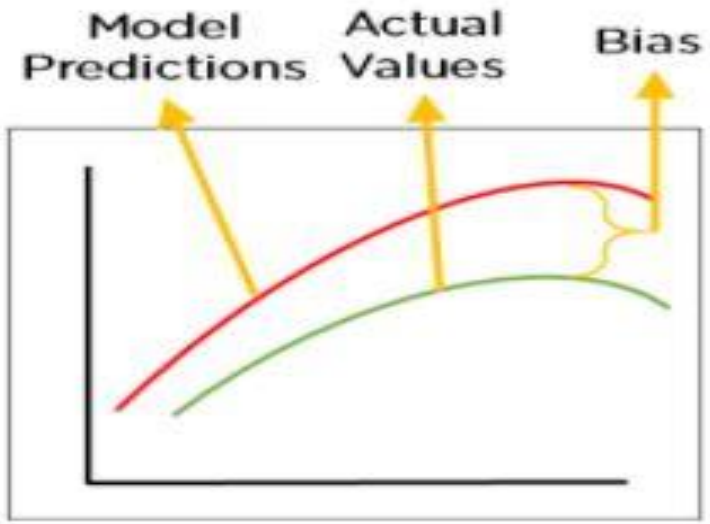


Bias and variance

What is Bias?

- Bias refers to the error that is introduced by approximating a real-world problem with a simplified model.
- A model with high bias is typically too simple and makes assumptions that do not true for the real-world problem.
- This can lead to underfitting, **where the model is unable to capture the underlying patterns** in the data and performs poorly on both the training and testing data.

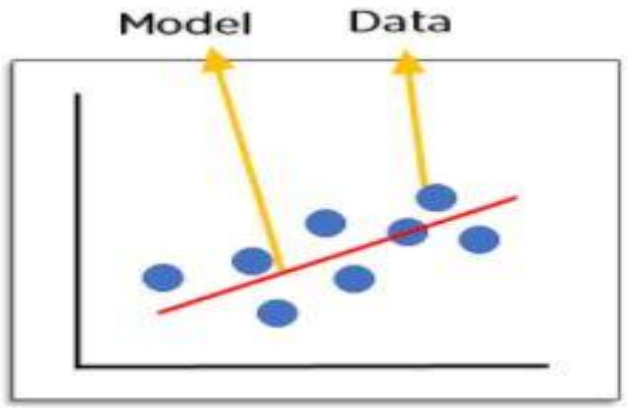
Bias is the difference between our actual and predicted values. Bias is the **simple assumptions** that our model makes about our data to be able to predict new data.



Bias and variance

- When the **Bias is high**, assumptions made by our model are too basic, the model **can't capture the important features of our data.**
- This means that our model **hasn't captured patterns in the training data** & hence cannot perform well on the testing data too.
- If this is the case, **our model cannot perform on new data .**
- This instance, where the **model cannot find patterns in our training set** and hence fails for both seen and unseen data, is called **Underfitting**

The below figure shows an example of Underfitting. As we can see, the **model has found no patterns in our data** and the line of best fit is a straight line that does not pass through any of the data points. The model has failed to train properly on the data given and cannot predict new data either.



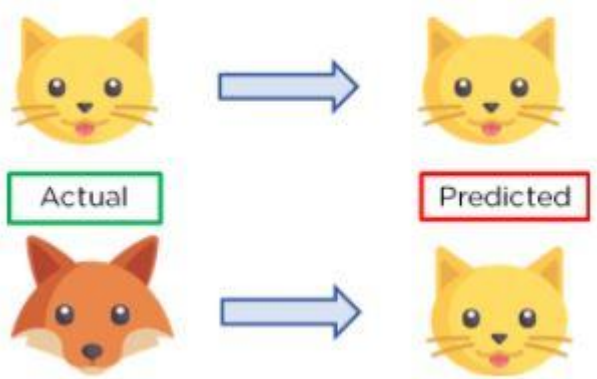
Bias and variance

What is Variance?

- Variance, on the other hand, refers to the amount by which the predictions of a model would change if it were trained on a different set of data.
- A model with high variance is able to **capture the noise or random fluctuations** in the training data and tends to **overfit**.
- This can result in good performance on the training data **but poor performance on new, unseen data**.

Bias and variance

What is Variance?

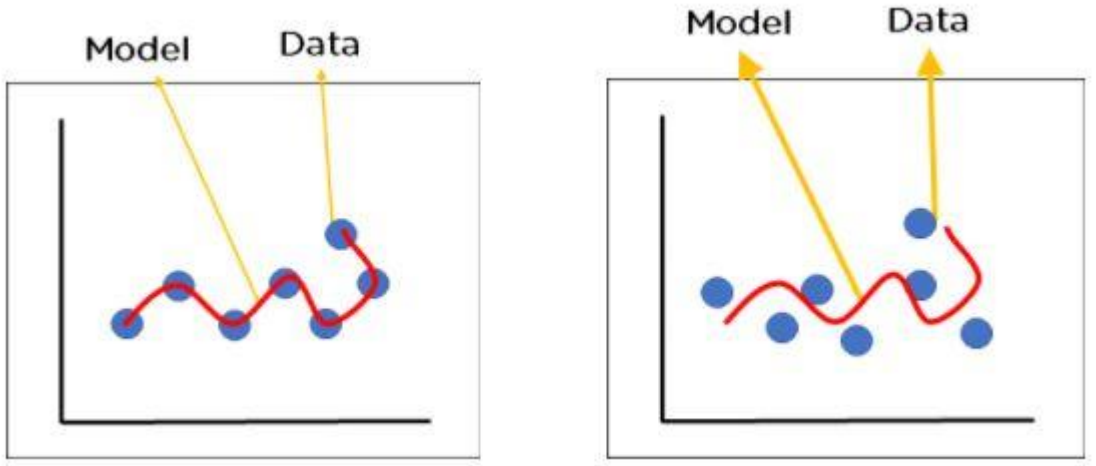


- we can see that our model has learned extremely well for our training data, which has taught it to identify cats.
- But when given new data, such as the picture of a fox, our model predicts it as a cat, as that is what it has learned.
- This happens when the Variance is high, our model will capture **all the features of the data given to it, including the noise**, will tune itself to the data, and predict it very well **but** when given new data, it cannot predict on it as it is too specific to training data.

.

Bias and variance

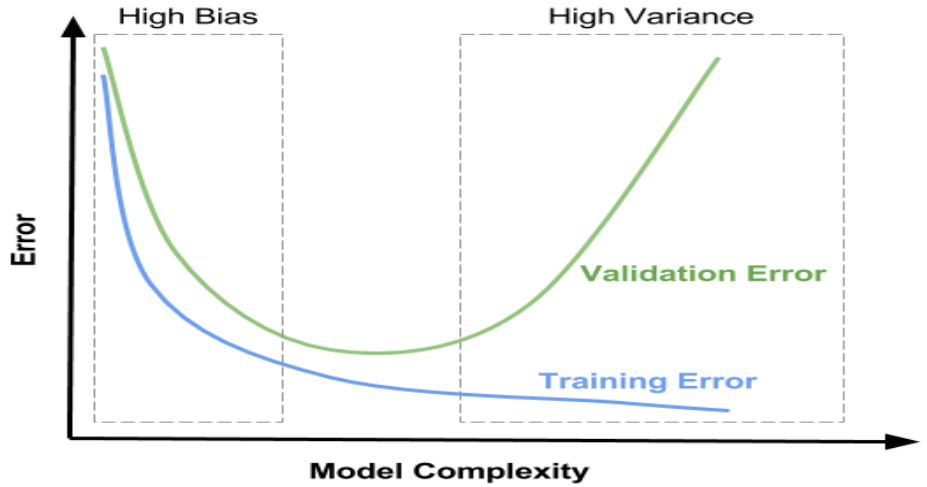
- Hence, model will perform really well on testing data and get high accuracy but will fail to perform on new, unseen data.
- New data may not have the exact same features and the model won't be able to predict it very well. **This is called Overfitting.**



Over-fitted model where we see model performance on, a) training data b) new data

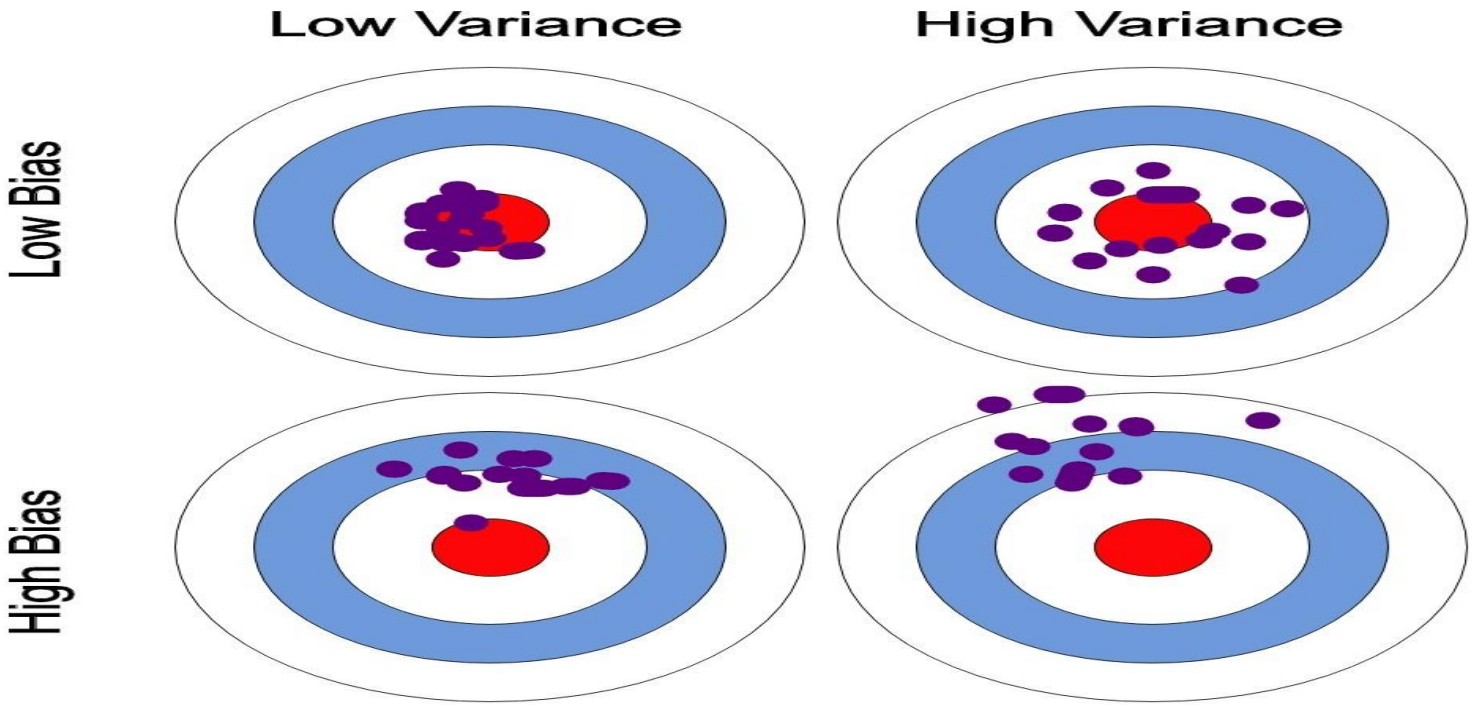
Bias-Variance Tradeoff

- In general, the goal of model **training** is to achieve a balance between bias and variance.
- This is often referred to as the **bias-variance tradeoff**.



- In the figure, we can see that when **bias is high**, the error in both testing and training set is also high.
- If we have a high variance, the model performs well on the training set, we can see that the error is low, but gives high error on the testing set.
- We can see that there is a region in the middle, where the error in both training and testing set is low and the bias and variance is in perfect balance.

Bias-Variance Tradeoff



- The above bull's eye graph helps explain bias and variance tradeoff better.
- The best fit is when the data is concentrated in the center, ie: at the bull's eye. We can see that as we get farther and farther away from the center, the error increases in our model.
- The best model is one where bias and variance are both low .

Bias-Variance Tradeoff

- A model with **high bias and low variance** may be too simple and fail to capture the complexity of the underlying data, underfit.
- While a model with **low bias and high variance** may overfit the training data and fail to generalize to new data.
- To improve the performance of a model, techniques such as **regularization, cross-validation, and ensemble methods** can be used to reduce bias and variance and find the optimal balance between the two.

Bias-Variance Tradeoff

- In machine learning, bias & variance are two sources of errors that can affect the performance of a model.
- Bias refers to the error introduced by the model's assumptions about the data, while variance refers to the error introduced by the model's sensitivity to fluctuations in the data.

To minimize bias and variance in a machine learning model, the following approaches can be taken:

- **Increase the complexity of the model**: If the model is too simple, it may have high bias and low variance. Increasing the complexity of the model by adding more features or layers can help reduce bias.
- **Regularization**: Regularization techniques such as L1, L2, or dropout can be used to reduce the complexity of the model and prevent overfitting.
- **Feature selection**: Feature selection techniques can be used to select the most relevant features for the model, which can help reduce the noise in the data and improve the accuracy of the model.

Bias-Variance Tradeoff

- **Cross-validation**: Cross-validation can be used to evaluate the performance of the model and identify the optimal hyperparameters.
- **Ensemble methods**: Ensemble methods such as bagging, boosting, or stacking can be used to combine multiple models and reduce the variance of the model.
- **Data augmentation**: Data augmentation techniques can be used to increase the size & diversity of the training data, which can help reduce overfitting and improve the accuracy of the model.

Overall, minimizing bias and variance in a machine learning model requires a **balance between simplicity and complexity,** as well as careful selection of features and regularization techniques to prevent overfitting

MIT-WPU

।। विश्वशान्तिर्ध्रुवं ध्रुवा ।।

Ensemble Method

Ensemble Learning

- Ensemble Learning is a technique that create multiple models and then combine them them to produce improved results.
- Ensemble learning usually produces more accurate solutions than a single model would.

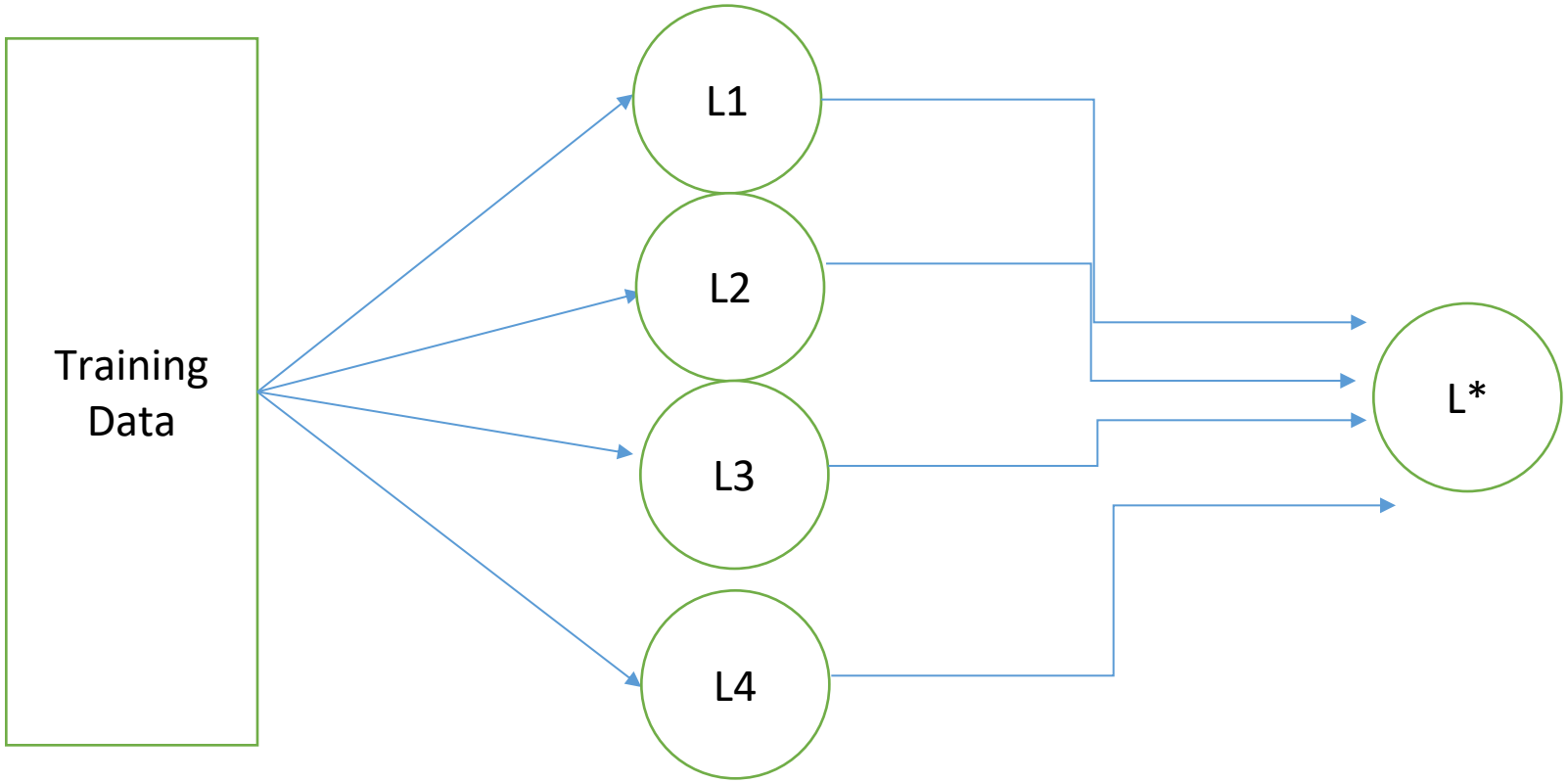
Ensemble Learning

- Ensemble learning methods is applied to regression as well as classification.
- Ensemble learning for regression creates multiple repressors i.e. multiple regression models such as linear, polynomial, etc.
- Ensemble learning for classification creates multiple classifiers i.e. multiple classification models such as logistic, decision tress, KNN, SVM, etc.

Ensemble Learning

- There are two steps in ensemble learning:
  - Multiples machine learning models were generated using same or different machine learning algorithm. These are called "base models".
  - The prediction perform on the basis of base models.

Ensemble Method



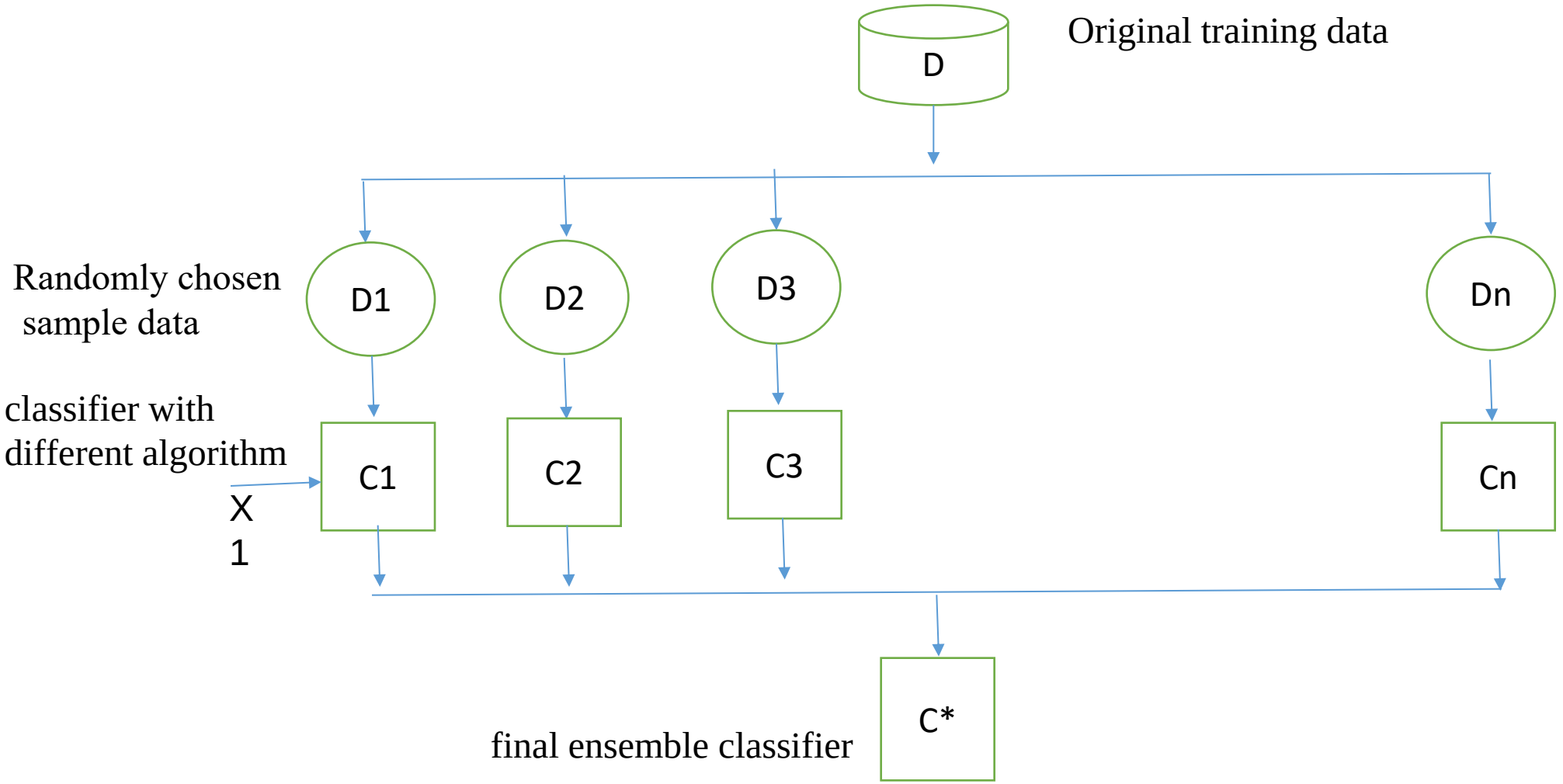
Ensemble Learning:

Multiple **Machine Learning** models (classifiers) are combined to solve a particular problem

Ensemble Method

- "A group of item viewed **as a whole** rather than individually"
- Ensemble method : not depend on just one model or algorithm for your output. Rather consider many model at same time.
- L1,L2,L3, L4……..Ln : **Base learner** with different classification algorithm like decision tree, KNN, naive bias, Linear regression, Logistic Regression…etc
- This situation is called as heterogeneous situation / Learner
- If you keep all L with same algorithm you can change data set and **assign different data set to every algorithm** or L to gain heterogeneous behavior
- L* : Final strong learner or model
- L*: Resultant classifier output of all Lerner must have more predictive power so called as strong classifier.

Ensemble Method



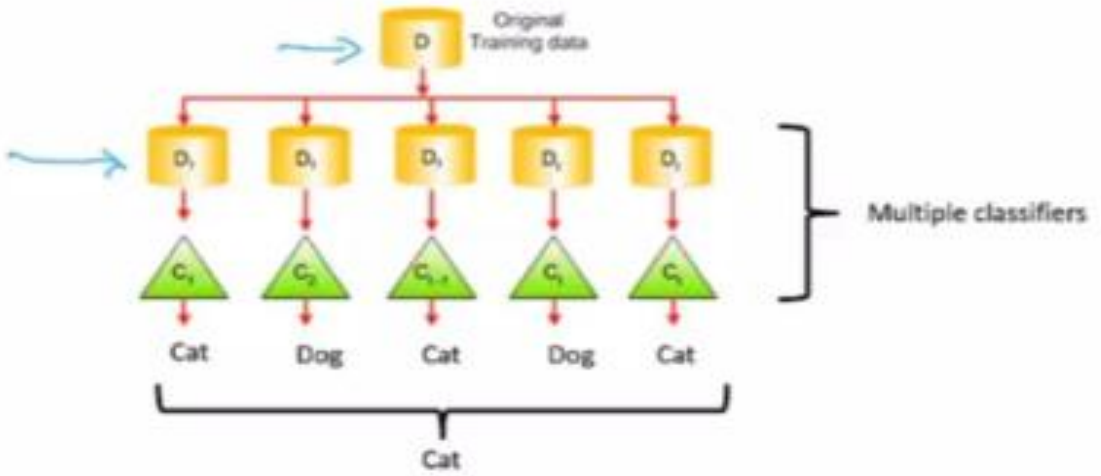
- Techniques/Methods
  - Voting and Averaging
  - Stacking
  - Bootstrap Aggregating/Bagging
  - Boosting

Ensemble Learning Techniques/Methods

- Voting and Averaging
  - Voting used in classification
  - Averaging used in regression

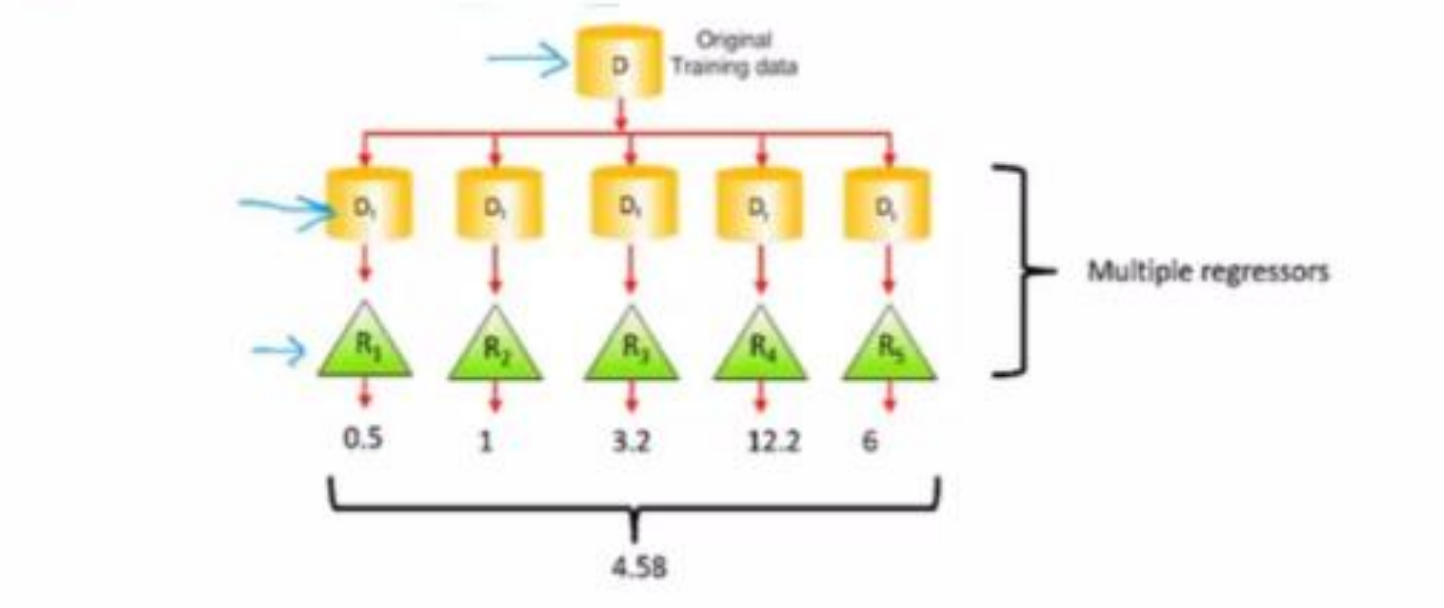
Ensemble Learning Techniques/Methods

- Voting and Averaging
  - Voting used in classification



Ensemble Learning Techniques/Methods

- Voting and Averaging
  - Averaging used in Regression

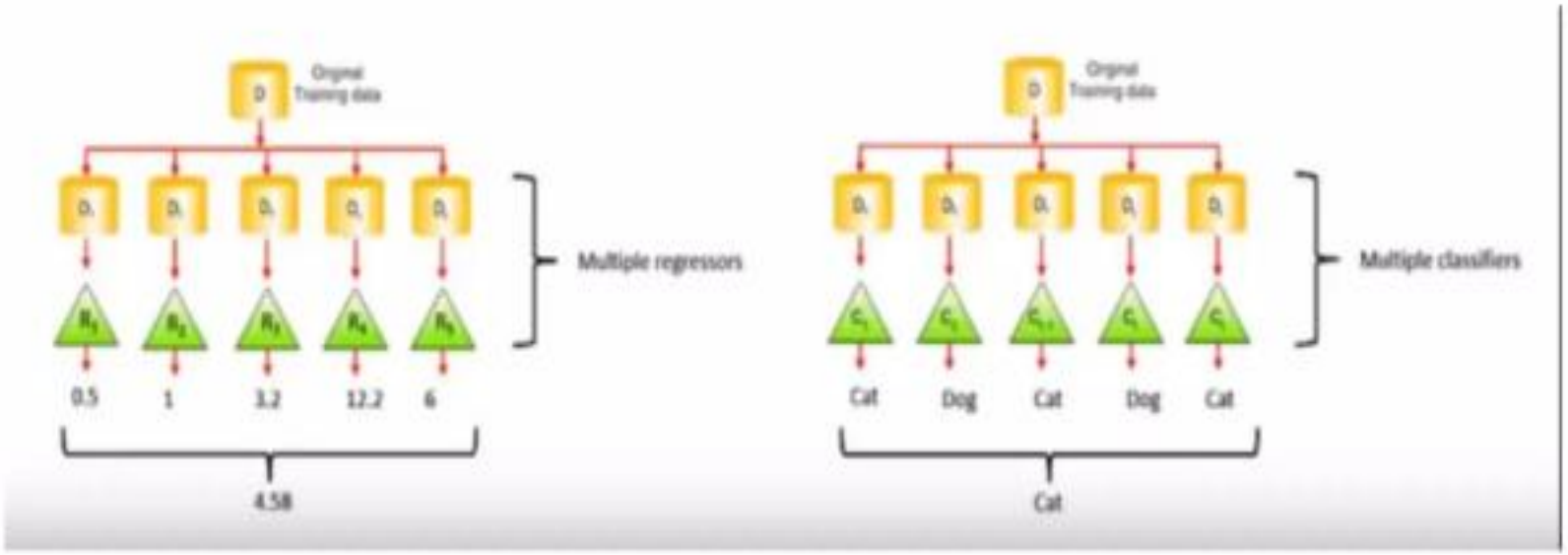


Ensemble Learning Techniques/Methods

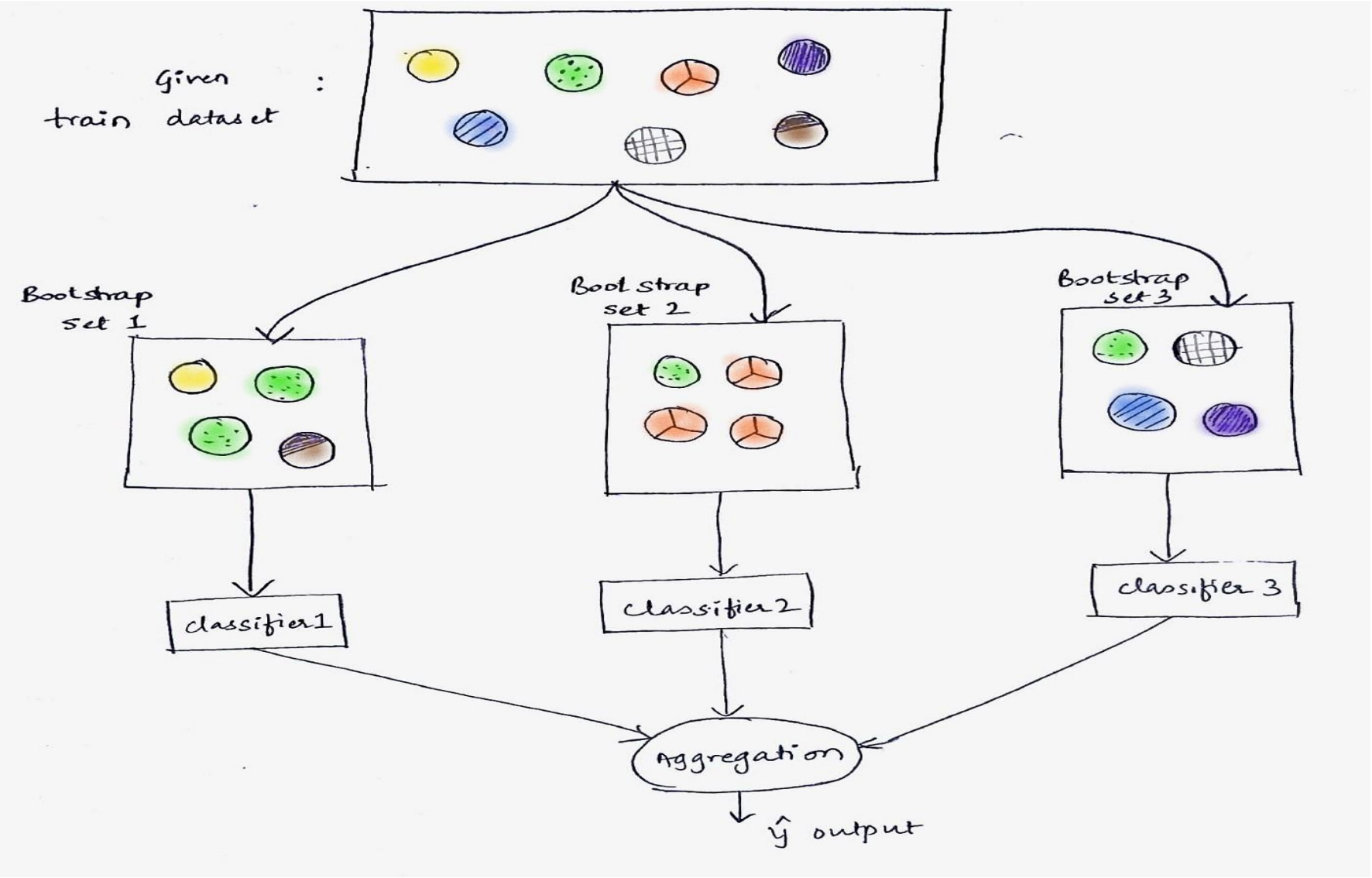
- Bootstrap Aggregating / Bagging
  - Again at first step multiple machine learning models are generated. These models are generated using the same machine learning algorithm with n random observations of m sub-samples of the original dataset using bootstrap sampling method.
  - The second step is aggregating the result generated from these models. Well known methods, such as voting and averaging, are used for this purpose.
  - For example, Random Forest algorithm uses the bagging technique.

Ensemble Learning Techniques/Methods

- Bootstrap Aggregating / Bagging
  - Voting and averaging is used in second step for final prediction



Bootstrap aggregation (Bagging)



Bootstrap aggregation (Bagging)

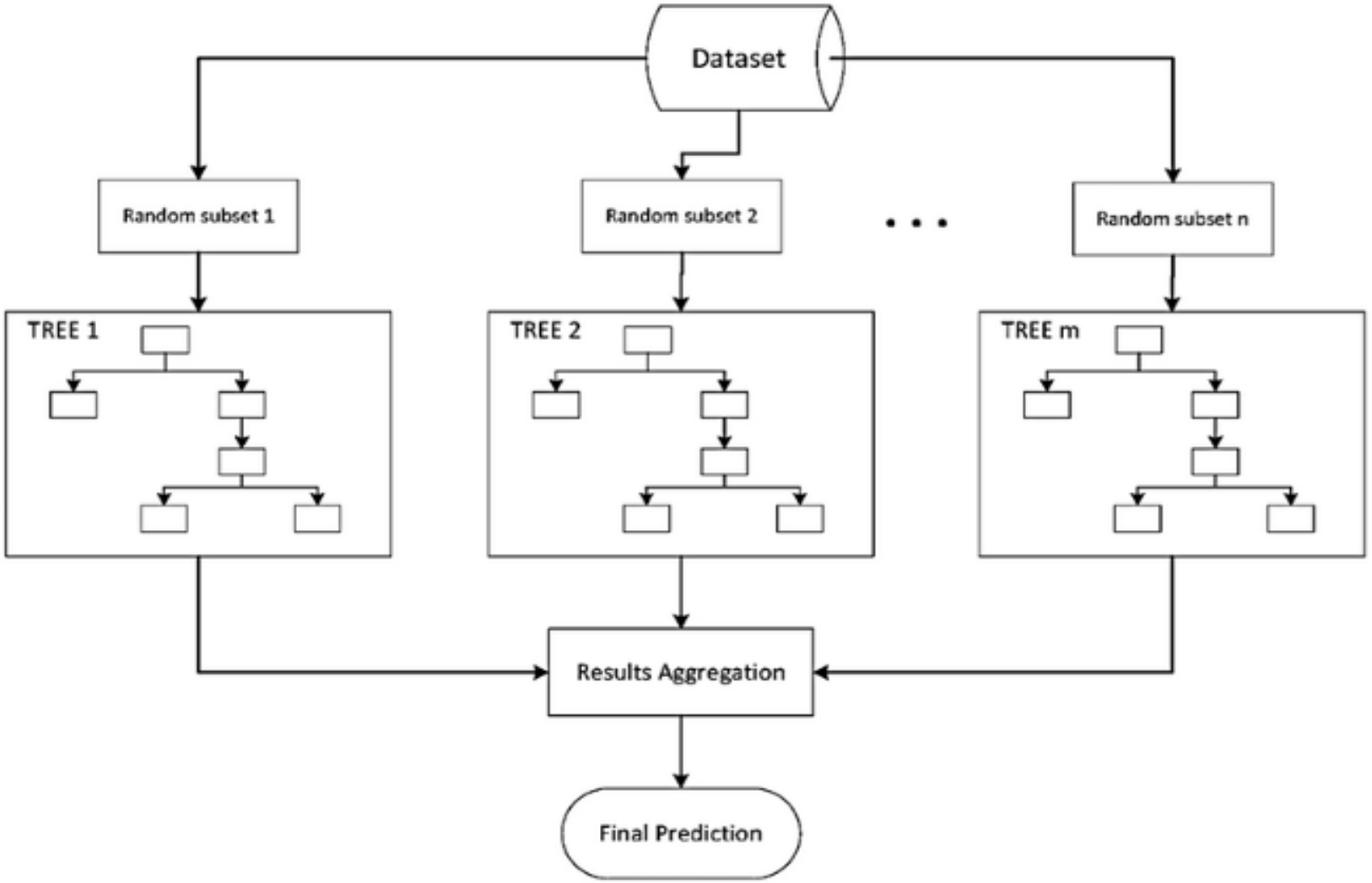
- D: Original training data(contains many sample, records)
- D1: Randomly chosen sample data record to make new dataset, **with replacement** (any one record may present repeatedly in records d2 d3 etc…i.e. multiple copies, or may be only one record is present in any data set and not in any other data)
- They are also called as **bootstrap sample data set, to train model** as c1 …cn classifier
- C1: classifier with different algorithm , or model, weak learner, classifier
- C* : **final ensemble classifier or model**, as strong classifier **because accuracy , predictive power, error rate less,** so result of this classifier will be more accurate precision

Bagging

- Reduces overfitting (variance)
- Normally uses one type of classifier
- Decision trees are popular
- Easy to parallelize

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

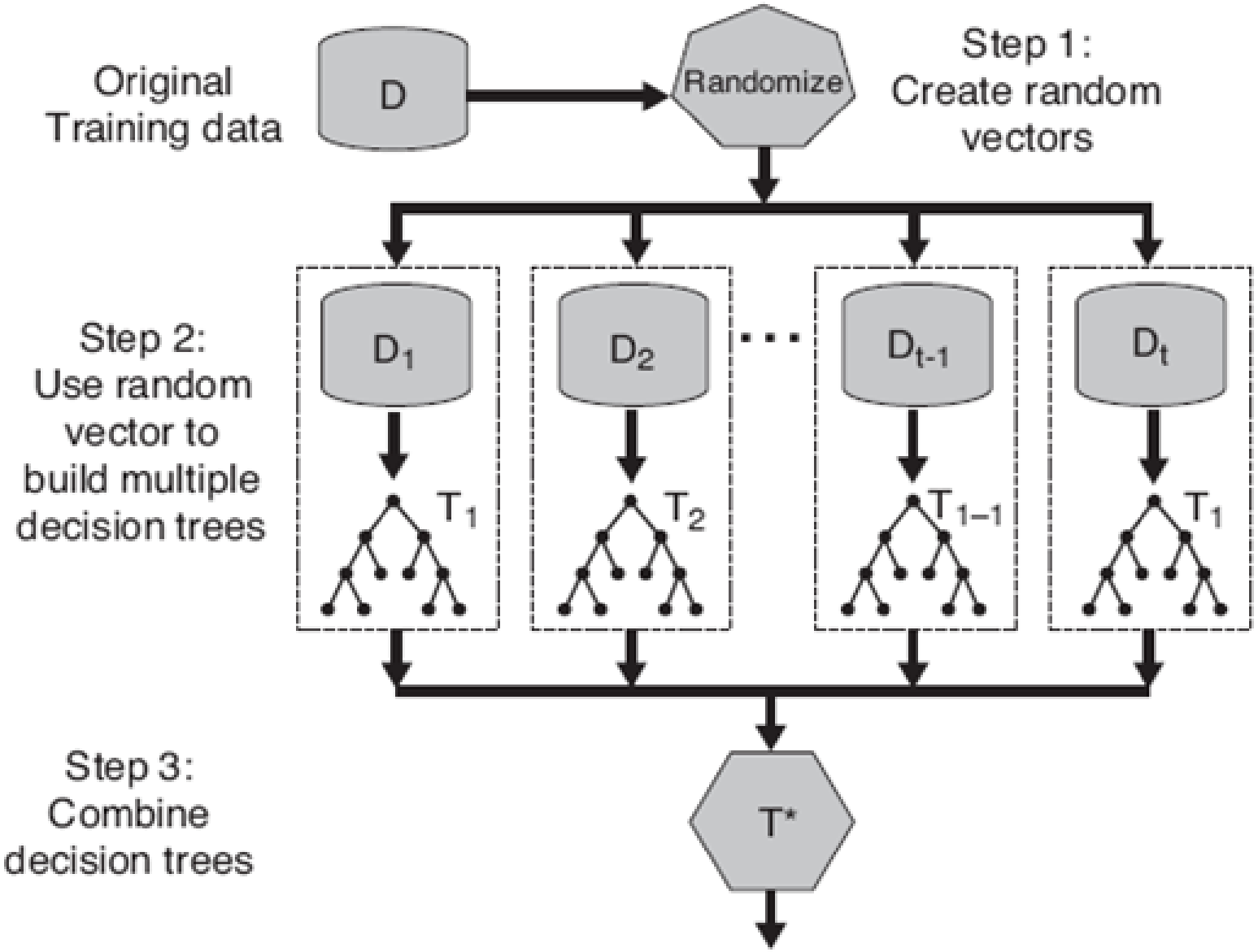
Bagging (Random Forest)



Random Forest Algorithm

- **Ensemble method specifically designed for** decision tree classifiers
- Random Forests grows many trees
  - Ensemble of unpruned decision trees
  - Each base classifier classifies a "new" vector of attributes from the original data
  - **Final result on classifying a new instance**: voting.
  - **Forest chooses the classification** result having the most votes (over all the trees in the forest)

RF

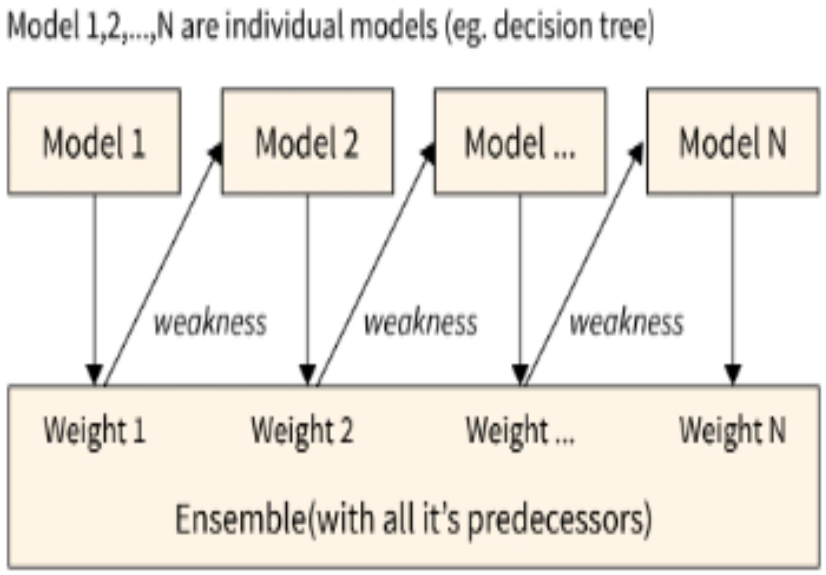

Original
Training data
D
Randomize
Step 1:
Create random
vectors
Step 2:
Use random
vector to
build multiple
decision trees
D1
D2
Dt-1
Dt
T1
T2
T1–1
T1
Step 3:
Combine
decision trees
T*

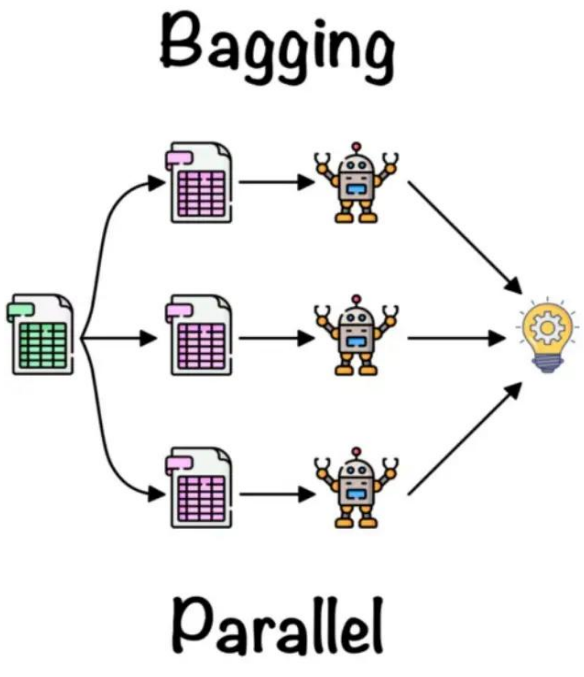
Ensemble Learning Techniques/Methods

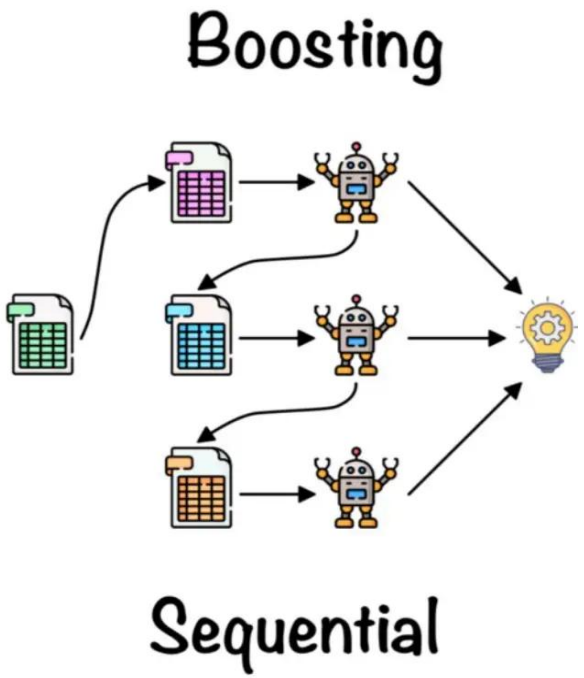
- Boosting
  - The term "boosting" is used to describe a family of algorithms which are able to convert weak models to strong models.
  - Boosting incrementally builds an ensemble by training each model with the same dataset but where the weights of instances are adjusted according to the error of the last prediction
  - Adaboost is a widely known algorithm which is a boosting method.

Boosting

- Boosting is an ensemble learning method that combines a set of weak learners into **strong learners** to minimize training errors.
- In boosting, a random sample of data is selected, fitted with a model, &then trained sequentially.
- That is, **each model** tries to **compensate for the weaknesses of its predecessor**.
- Each classifier's weak rules are combined with each iteration to form one strict prediction rule.







MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

Boosting

- In boosting, the weak learners are typically decision trees, but they can also be other types of classifiers, such as linear models or neural networks.
- During the training process, the weak learners are assigned weights based on their performance.
- Those that perform well are given higher weights, while those that perform poorly are given lower weights.
- When **predicting the class label** of a **new instance**, the outputs of all the weak learners are combined in a weighted sum, with the weights determined by their performance during training.
- The final output is the sign of the weighted sum, which can be interpreted as the probability that the instance belongs to the positive class.

Boosting

Here's how the algorithm works:

- Step 1: The base algorithm reads the data and assigns equal weight to each sample observation.

- Step 2: False predictions made by the base learner are identified. In the next iteration, these false predictions are assigned to the next base learner with a higher weightage on these incorrect predictions.

- Step 3: Repeat step 2 until the algorithm can correctly classify the output.

Boosting

Some popular boosting algorithms include

1. AdaBoost (Adaptive Boosting),
2. Gradient Boosting,
3. XGBoost.

- **These algorithms** differ in the way the weights are updated during training and the **loss function** used to measure performance.
- Boosting has been shown to be highly effective in many classification and regression tasks, and is widely used in practice..

AdaBoost (Adaptive Boosting)

- AdaBoost, short for Adaptive Boosting, is a machine learning algorithm used for classification and regression tasks.
- It is an ensemble method that combines several "weak" learners (i.e., classifiers that perform only slightly better than random guessing) into a "strong" ensemble classifier.

- The basic idea of AdaBoost is to iteratively train weak classifiers on the training data
- while adjusting the weights of the training examples to emphasize the examples that were misclassified by the previous weak classifier.
- The final strong classifier is a **weighted sum of the weak classifiers**, where each weak classifier is assigned a weight proportional to its performance.

AdaBoost (Adaptive Boosting)

Here are the steps involved in AdaBoost:

- Initialize the weights of the training examples to be equal.
- Train a weak classifier on the training data. A weak classifier is a classifier that performs only slightly better than random guessing.
- Evaluate the performance of the weak classifier on the training data using a weighted error rate. The weighted error rate is the sum of the weights of the misclassified examples divided by the sum of all weights.
- Update the weights of the training examples based on the performance of the weak classifier. Increase the weights of the misclassified examples and decrease the weights of the correctly classified examples.
- Repeat steps 2-4 for a fixed number of iterations or until the training error reaches a certain threshold.
- Combine the weak classifiers into a final strong classifier using a weighted sum of the weak classifiers, where each weak classifier is assigned a weight proportional to its performance

AdaBoost (Adaptive Boosting)

The most important parameters are

base_estimator, n_estimators, and learning_rate

- base_estimator: It is a weak learner used to train the model.

It uses **DecisionTreeClassifier as default** weak learner for training purpose.

You can also specify different machine learning algorithms.

- n_estimators: Number of weak learners to train iteratively.

- learning_rate: It contributes to the weights of weak learners. It uses 1 as a default value.

AdaBoost (Adaptive Boosting)

Pros

- AdaBoost is easy to implement.
- It iteratively corrects the mistakes of the weak classifier and improves accuracy by combining weak learners.
- You can use many base classifiers with AdaBoost. AdaBoost is not prone to overfitting.

Cons

- AdaBoost is sensitive to noise data.
- It is highly affected by outliers because it tries to fit each point perfectly.
- AdaBoost is slower compared to XGBoost.

Gradient Boosting

- Gradient Boosting is an ensemble learning technique that **combines multiple weak learners** (usually decision trees) **to build a strong model**.

- The idea behind gradient boosting is to **iteratively add new trees to the model**, each tree correcting the errors made by the previous ones.

- **During each iteration**, the algorithm fits a new tree to the residual errors of the previous model.
- Gradient Boosting is particularly useful when dealing with complex datasets and non-linear relationships between variables.

Gradient Boosting

Here are some of the most important parameters:

- n_estimators: This parameter controls the number of trees in the ensemble.
- learning_rate: This parameter controls the contribution of each tree to the final prediction.
- max_depth: This parameter controls the maximum depth of each tree in the ensemble. Increasing this parameter can improve the model's performance, but may also increase the risk of overfitting.
- loss: This parameter controls the loss function used to measure the quality of each split in the trees. s

XGBoost (Extreme Gradient Boosting)

- XGBoost (Extreme Gradient Boosting) is a decision tree-based ensemble machine learning algorithm that uses gradient boosting to iteratively improve the performance of a model.
- XGBoost has several advantages over traditional gradient boosting algorithms.
- It includes a number of regularization techniques to prevent overfitting, making it more accurate and less prone to overfitting on training data.
- It also includes a more efficient implementation of gradient boosting, making it faster and more scalable.

XGBoost (Extreme Gradient Boosting)

Here's how it works:

- **Initialization**: The algorithm starts with a single decision tree, which serves as the **initial model**. The output of this tree is used to make predictions on the training data.
- **Calculation of Residuals**: The difference between the predicted values and actual values for the training data is calculated, which is called the residuals. These residuals are used to train the next decision tree in tl ensemble.
- **Tree Construction**: A new decision tree is constructed to predict the residuals calculated in the previous step. **This tree is fitted on the residuals** instead of the actual target variable.

The tree is constructed in a greedy manner by recursively splitting the data into smaller subsets based on the feature that provides the most information gain.

- **Update Model**: The output of the new decision tree is **combined** with the output of the previous tree to produce a new set of predictions. This updated model is used to calculate the residuals for the next iteratio
- **Iteration**: Steps 3 and 4 are repeated iteratively until the residuals can no longer be reduced **or** a pre-defined maximum number of trees is reached. The final model is the sum of all the individual tree models.

Comparison of Boosting Algorithms

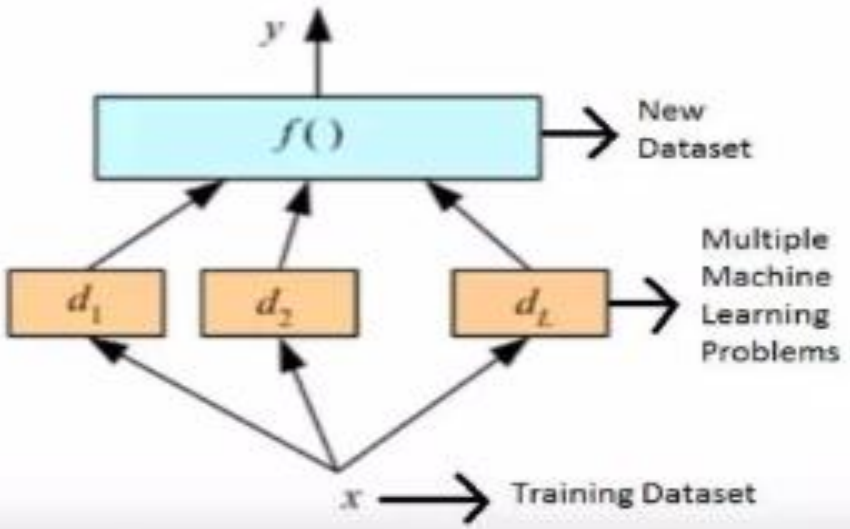
Feature	AdaBoost	Gradient Boosting	XGBoost
Type of algorithm	Boosting	Boosting	Boosting
Type of weak learners	Decision trees, SVM, etc.	Decision trees	Decision trees
Training process	Sequentially adjusting weights of examples	Iteratively adding weak models to the ensemble	Iteratively adding weak models to the ensemble
Residuals	Weighted sum of errors	Gradient of loss function	Gradient of loss function
Handling outliers	Sensitive	Robust	Robust
Handling missing values	No	No	Yes
Regularization	No	Yes	Yes
Parallel processing	No	No	Yes
Hyperparameter tuning	Fewer	More	More
Speed	Slow	Medium	Fast
Accuracy	Medium	High	Very high
Ease of implementation	Easy	Medium	Medium
Used in	Real-world applications	Real-world applications	Real-world applications

Ensemble Learning Techniques/Methods

- Stacking
  - Also known as stacked generalization
  - Multiple machine learning models are combined using another (combiner) machine learning algorithm.
  - The basic idea is to train machine learning algorithms with training dataset and then generate a new dataset with these models. Then this new dataset is used as input for the combiner machine learning algorithm.

Ensemble Learning Techniques/Methods

- Stacking



Stacking

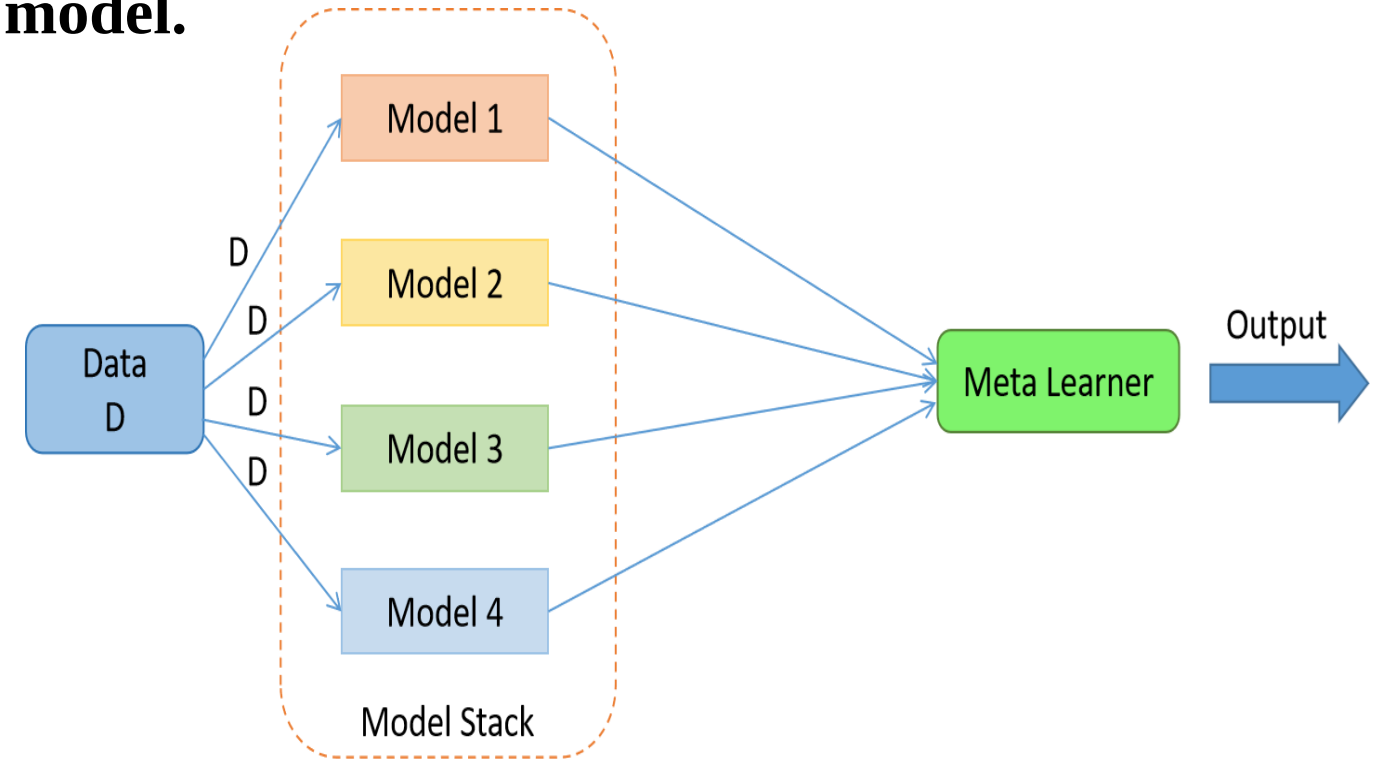
- It is an ensemble method that combines multiple models (classification or regression) via meta-model (meta-classifier or meta-regression).
- The base models are trained on the complete dataset, then the meta-model is trained on features returned (as output) from base models.
- The base models in stacking are typically different.
- The meta-model helps to find the features from base models to achieve the best accuracy.

Stacking

- Stacking is a bit different from the basic ensembling methods because it has first-level and second-level models.
- Stacking features are first extracted by training the dataset with all the first-level models.
- A first-level model is then using the train stacking features to train the model than this model predicts the final output with test stacking features.

Stacking

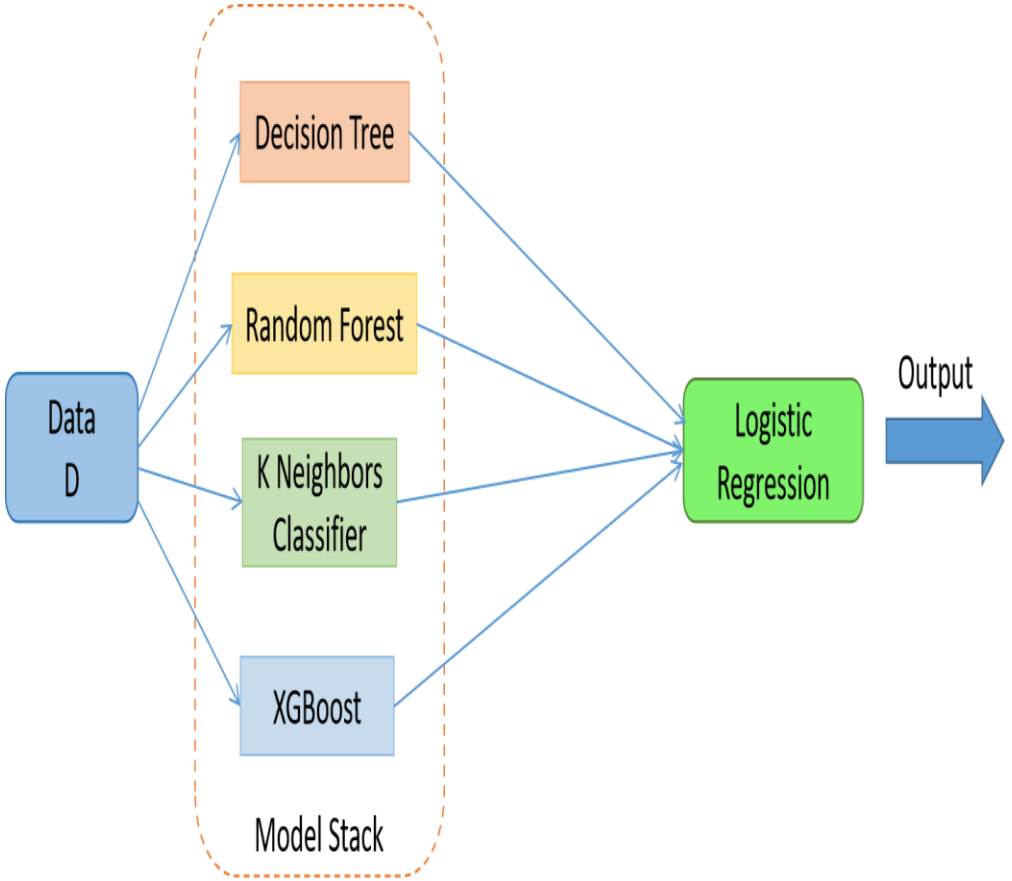
- Stacking is a popular **ensemble learning technique** used in machine learning to improve the predictive performance of models.
- The idea behind stacking is to **combine several base models**, also known as level-0 models, to **form a new model**, known as a level-1 or meta model.
- The **base models are trained on the same dataset**, and their outputs are then **used as input features to the meta model.**



Stacking

The stacking process involves the following steps:

- Splitting the training data into two or more folds.
- Training several base models on each fold of the training data.
- Using the base models to make predictions on the validation data, which was not used during training.
- Using the predictions from the base models as features to train a meta model on the validation data.
- Repeating steps 2-4 for each fold of the training data.
- Once the meta model has been trained on the validation data, it can be used to make predictions on new, unseen data.



Stacking

- Stacking can improve the predictive performance of models because it combines the strengths of different models.
- For example, one base model may be good at predicting certain patterns in the data, while another model may be better at predicting other patterns.
- By combining the predictions from multiple models, the meta model can make more accurate predictions than any individual model.

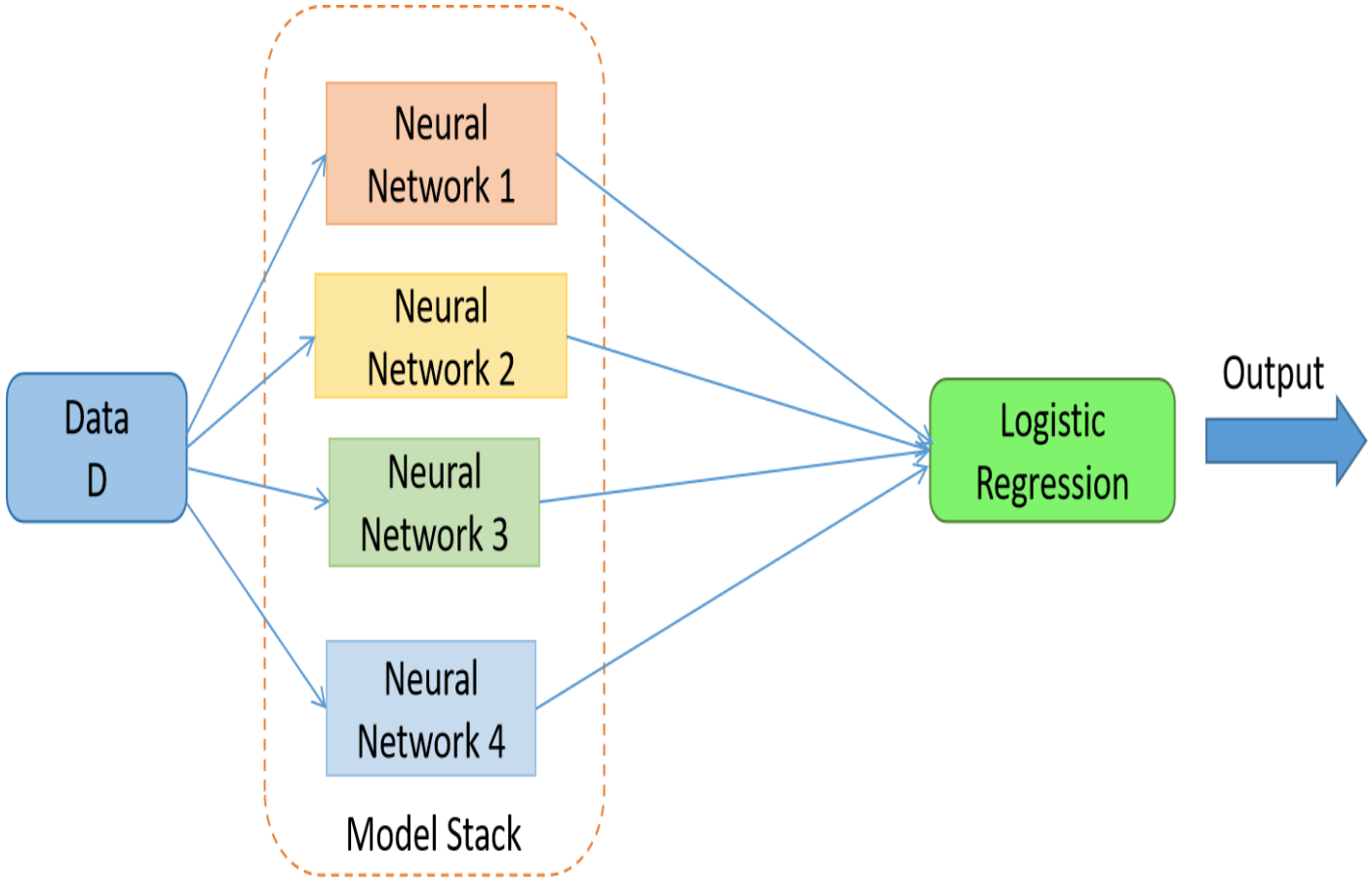
➢ One potential downside of stacking is that it can be computationally expensive, especially if the base models are complex or if the dataset is large.

➢ Additionally, if the base models are overfitting the training data, then the stacking ensemble may also overfit the data.

To mitigate this, it is important to use a diverse set of base models and to tune the hyperparameters of each model carefully.

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

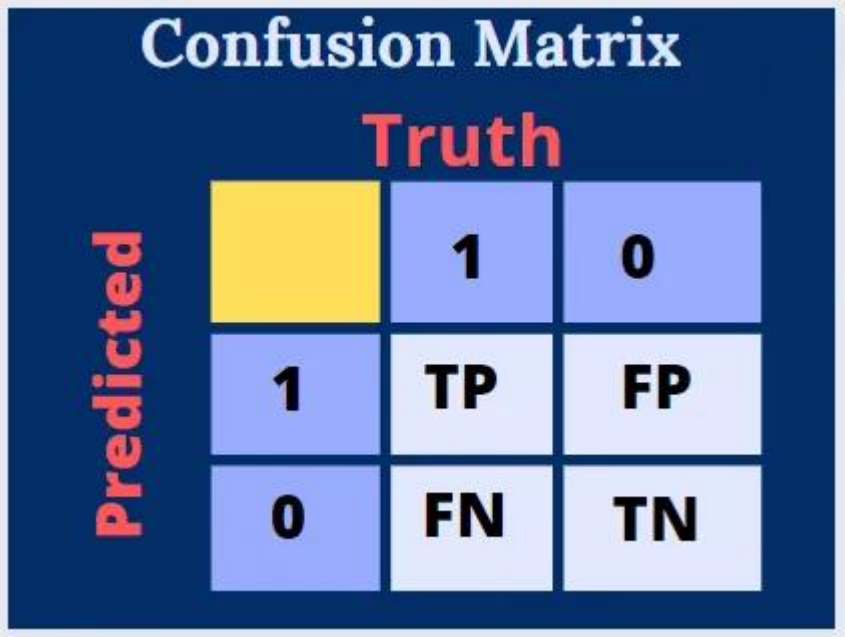
Stacking for Deep Learning



How to evaluate ML models

Classification metrics: confusion matrix

A confusion matrix is a table that is often used to evaluate the performance of a classification model. It displays the number of true positives, false positives, true negatives, and false negatives in a tabular format. Here's an example of a confusion matrix for a binary classification problem:



How to evaluate ML models

confusion matrix : example of a pregnancy test, where an actual pregnant woman and a fat man consult a doctor, and the test results are given in the below image.:

How to evaluate ML models

TP(True Positive): The woman is pregnant, and she is predicted as pregnant. Here P represents positive prediction, and T shows that our prediction is actually true.

FP(False Positive): A fat man is predicted as pregnant, which is actually false. Here P represents positive prediction, and F shows that our prediction is actually false. This is also called a Type I error.

FN(False Negative): A woman who is actually pregnant is predicted as not pregnant. Here N represents negative prediction, and F shows that our prediction is actually false. This is also called a Type II error.

TN(True Negative): A fat man is predicted as not pregnant. Here N represents Negative prediction, and T shows that our prediction is actually true.

Accuracy

Accuracy : is the simplest metric and can be defined as the number of test cases correctly classified divided by the total number of test cases. A

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

- It can be applied to most generic problems but is not very useful when it comes to unbalanced datasets.
- For instance, if we are detecting frauds in bank data, the ratio of fraud to non-fraud cases can be 1:99. In such cases, if accuracy is used, the model will turn out to be 99% accurate by predicting all test cases as non-fraud. The 99% accurate model will be completely useless.
- If a model is poorly trained such that it predicts all the 1000 (say) data points as non-frauds, it will be missing out on the 10 fraud data points. If accuracy is measured, it will show that that model correctly predicts 990 data points and thus, it will have an accuracy of (990/1000)*100 = 99%!
- This is why accuracy is a **false indicator** of the model's health. Therefore, for such a case, a metric is required that can focus on the ten fraud data points which were completely missed by the model. Imbalance dataset ~~: Accuracy~~

Precision

Precision is the proportion of true positive predictions among all positive predictions made **by the classifier**.

Precision = TP / (TP + FP)

Precision is the metric used to identify the correctness of classification.

- Higher is the precision, which means better is the ability of the model to correctly classify the positive class.
- In the problem of **predictive maintenance** (where one must predict in advance when a machine needs to be repaired), precision comes into play.
- The cost of maintenance is usually high and thus, incorrect predictions can lead to a loss for the company.
- In such cases, the ability of the model to correctly classify the positive class and to lower the number of false positives is paramount!
- The ideal value of precision would be 1.0, which means that all of the positive predictions made by the model are correct.

Recall & F1 score

Recall (also known as sensitivity or **true positive rate-TPR**) is the proportion of true positive predictions among all positive instances in the dataset. It is calculated as: **Recall = TP / (TP + FN)**

True Positive Rate(TPR): True Positive/positive (actual +vs in dataset)

In fraud problem, the recall value will be very useful in fraud cases **because** a high recall value will indicate that a lot of fraud cases were identified out of the total number of frauds.

- The ideal value of recall would also be 1.0, which means that the model correctly identifies all positive cases in the dataset.

F1 score is the harmonic mean of precision and recall. It provides a single score that balances both precision and recall. It balances out the strengths of each

F1 score = 2 * (Precision * Recall) / (Precision + Recall)

- It is useful in cases where both recall and precision can be valuable – like in the identification of plane parts that might require repairing. Here, precision will be required to save on the company's cost (because plane parts are extremely expensive) and recall will be required to ensure that the machinery is stable and not a threat to human lives.

Confusion Matrix:

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Confusion Matrix-Example

Let's now define the most basic terms, which are whole numbers (not rates):

- **true positives (TP):** These are cases in which we predicted yes (they have the disease), and they do have the disease.
- **true negatives (TN):** We predicted no, and they don't have the disease.
- **false positives (FP):** We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
- **false negatives (FN):** We predicted no, but they actually do have the disease. (Also known as a "Type II error.")

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?
- Perfect score is 1.0

$$recall = \frac{TP}{TP + FN}$$

- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

TPR, FPR, TNR, FNR

Rate is a measure factor in a confusion matrix. It has also 4 type TPR, FPR, TNR, FNR

- True Positive Rate(TPR): True Positive/positive ..recall (True positives from all actual positive from dataset)

  True Positive Rate(TPR)=TP / (TP + FN)

- False Positive Rate(FPR): False Positive /**Negative** (false positives from all actual negative from dataset

  False Positive Rate(FPR) = FP/(FP+TN)

- False Negative Rate(FNR): False Negative/**Positive** (false negatives from all actual positive from dataset

  False Negative Rate(FNR)= FN/(FN+TP)

- True Negative Rate(TNR): True Negative/Negative (True negatives from all actual negative from dataset)

  True Negative Rate(TNR) = TN/(TN+FP)

For better performance, TPR, TNR should be high and FNR, FPR should be low

Confusion Matrix		Target		
		Positive	Negative	
Model	Positive	a	b	*Positive Predictive Value* a/(a+b)
	Negative	c	d	*Negative Predictive Value* d/(c+d)
		Sensitivity	*Specificity*	**Accuracy** = (a+d)/(a+b+c+d)

Confusion Matrix		Target			
		Positive	Negative		
Model	Positive	70 TP	20 FP	*Positive Predictive Value* ***precision***	0.78
	Negative	30 FN	80 TN	*Negative Predictive Value*	0.73
		*Sensitivity (**Recall**) (TPR)*	*Specificity (TNR)*	**Accuracy** = 0.75	
		0.70	0.80		

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\textbf{Specificity} = \frac{TN}{TN + FP}$$

Classification Rate/Accuracy

- Accuracy or classification accuracy tells the number of correct predictions made by the model.
- It is the ratio of the number of correct predictions to the total number of input samples.
- However, there are problems with accuracy. It assumes equal costs for both kinds of errors. A 99% accuracy can be excellent, good, mediocre, poor, or terrible depending upon the problem.

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Misclassification Rate (error)

- Accuracy or classification accuracy tells the number of wrong predictions made by the model.
- It is the ratio of the number of wrong predictions to the total number of input samples.

$$\text{Misclassification} = \frac{FP + FN}{TP + TN + FP + FN}$$

Precision / Positive Predictive Value (PPV)

- When it predicts yes, how often is it correct?
- Precision is defined as the number of true positives divided by the number of true positives plus the number of false positives. Precision is about being precise.
- Out of all the predictive positive classes, how much we predicted correctly

$$\text{Precision} = \frac{TP}{TP + FP}$$

Sensitivity / Recall / True Positive Rate

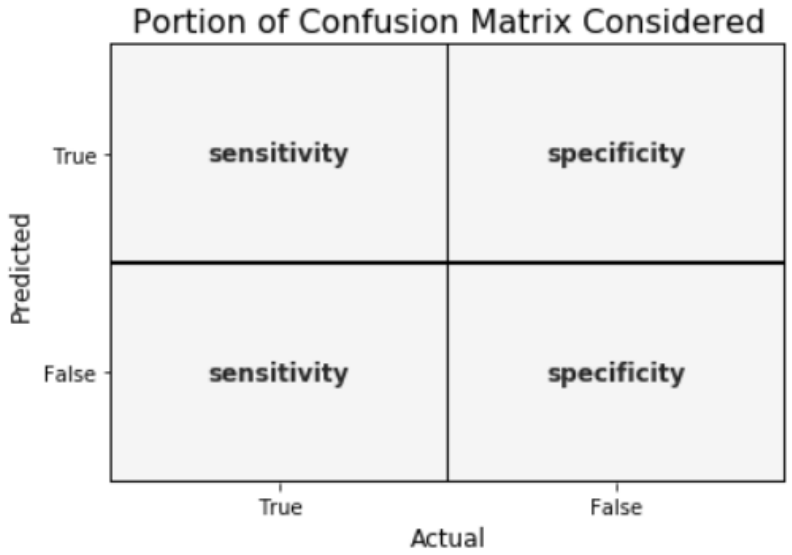
- When it's **actually yes, how often does it predict yes**?
- The recall is the fraction of positive events that were predicted correctly.
- 'When it is actually the positive result, how often does it predict correctly?'
- **High Recall** indicates the class is correctly recognized (a small number of FN).

$$\text{Recall} = \frac{TP}{TP + FN}$$

Specificity / True Negative Rate

- **When it's no, how often does it predict no?**
- It is the **True Negative Rate** or the proportion of true negatives to everything that should have been classified as negative.

$$Specificity = \frac{TN}{TN + FP}$$



Note: together, specificity and sensitivity consider the full confusion matrix.

F1 Score

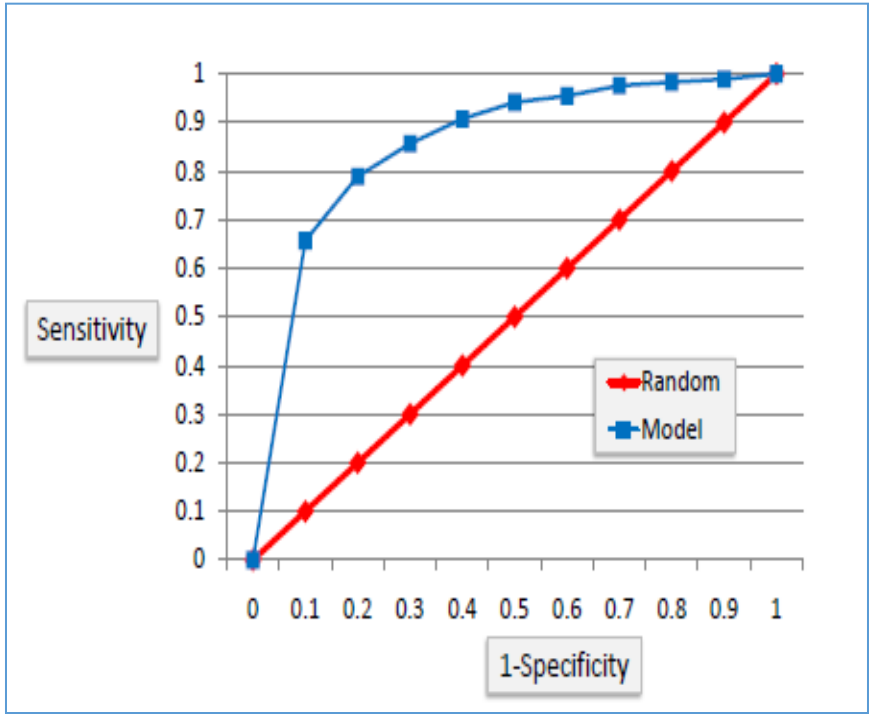
- The F1 score is the harmonic mean of the precision and recall, where
  - F1 score reaches its best value at 1 (perfect precision and recall)
  - F1 score reaches its worst value at 0

$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

- A good F1 score means (Low FP & Low FN): Correctly identifying real threats, and not disturbed by false alarms
- F1 is usually more useful than accuracy, **especially for an uneven class distribution**
- Harmonic mean instead of arithmetic mean: HM punishes extreme values more

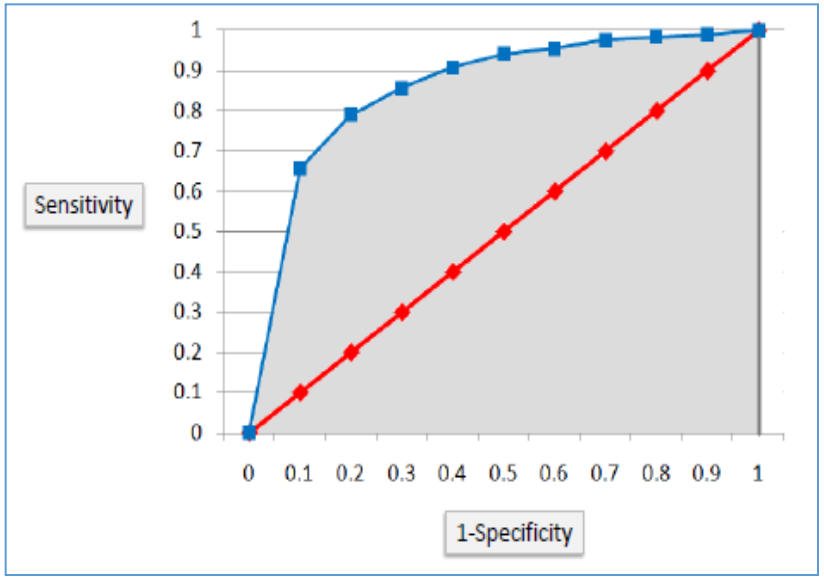
ROC Chart ..Receiver Operating Characteristic

- The ROC chart: false positive rate (FP**R**) (1-specificity) on X-axis, the probability of target=1 when its true value is 0
- Against the true positive rate(**TPR**)(sensitivity) on Y-axis, the probability of target=1 when its true value is 1
- Ideally, the curve will climb quickly toward the top-left meaning the model correctly predicted the cases.
- The diagonal red line is for a random model



Area Under the Curve (AUC)

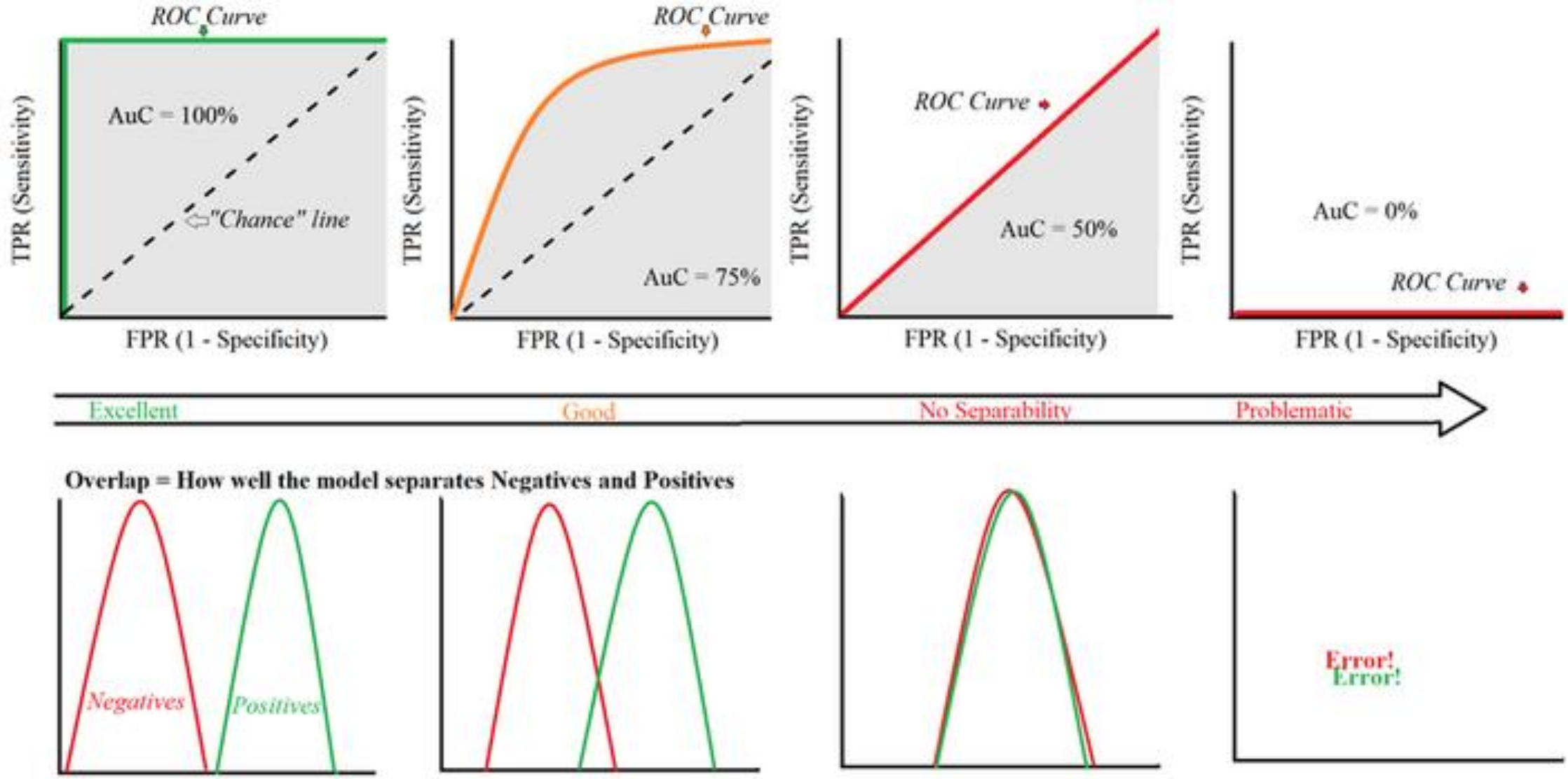
- Area under ROC curve is often used as a measure of quality of the classification models.
- A random classifier has an area under the curve of 0.5, while AUC for a perfect classifier is equal to 1.
- In practice, most of the classification models have an AUC between 0.5 and 1



ROC-AUC

- An area under the ROC curve of 0.8, for example, **means** that a randomly selected **case from the group with the target equals 1 has a score larger than** that for a **randomly chosen case from the group with the target equals 0** in 80% of the time.
- When a classifier cannot distinguish between the two groups, the area will be equal to 0.5 (the ROC curve will coincide with the diagonal).
- When there is a perfect separation of the two groups, i.e., no overlapping of the distributions, the area under the ROC curve reaches to 1 (the ROC curve will reach the upper left corner of the plot).

ROC curve

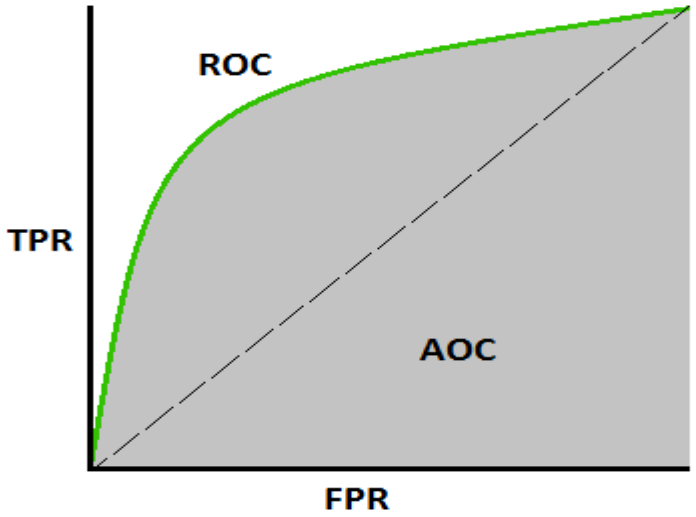

ROC Curve
AuC = 100%
"Chance" line
TPR (Sensitivity)
FPR (1 - Specificity)
ROC Curve
AuC = 75%
TPR (Sensitivity)
FPR (1 - Specificity)
ROC Curve
AuC = 50%
TPR (Sensitivity)
FPR (1 - Specificity)
AuC = 0%
ROC Curve
TPR (Sensitivity)
FPR (1 - Specificity)
Excellent
Good
No Separability
Problematic
Overlap = How well the model separates Negatives and Positives
Negatives
Positives
Error!
Error!

AUC-ROC

- ROC curve is a plot of true positive rate (recall) against false positive rate (TN / (TN+FP)).
- AUC-ROC stands for Area Under the **Receiver Operating Characteristics**
- Higher the area, the better is the model performance. Used for only binary classification.

If the curve is somewhere near the 50% diagonal line, it suggests that the model randomly predicts the output variable

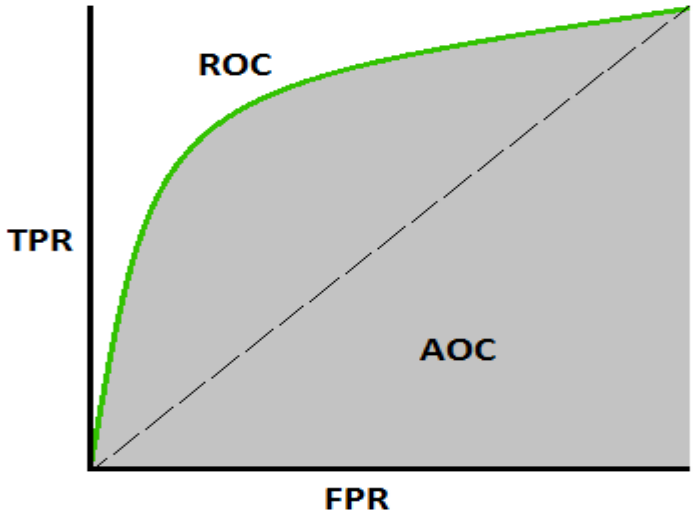
True Positive Rate(TPR)=TP / (TP + FN) **False positive Rate(TNR) = FP/(TN+FP)**



AUC - ROC Curve [Image 2] (Image courtesy: My Photoshopped Collection)

AUC-ROC

- The AUC ROC score is the area under the ROC curve and ranges from 0 to 1, with a score of 0.5 indicating a random classifier
- A score of 1.0 indicating a perfect classifier.
- A higher AUC ROC score indicates **better performance** of the model in distinguishing between positive and negative classes.



AUC - ROC Curve [Image 2] (Image courtesy: My Photoshopped Collection)

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?
- Perfect score is 1.0

$$recall = \frac{TP}{TP + FN}$$

- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

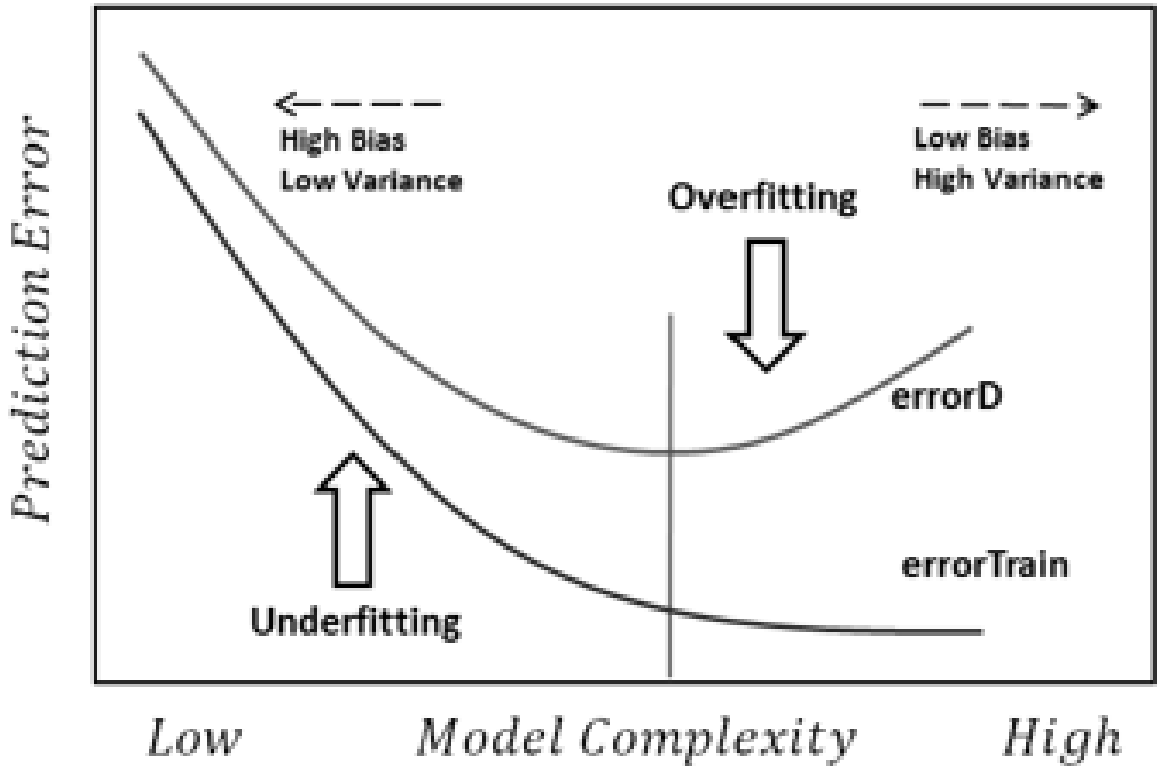
Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*
cancer = no	**140**	**9560**	9700	98.56 (*specificity)*
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Overfitting

- Natural end of process in DT is 100% purity in each leaf
- This **overfits** the data, which end up fitting noise in the data
- Overfitting leads to low predictive accuracy of new data
- Past a certain point, the error rate for the validation data starts to increase

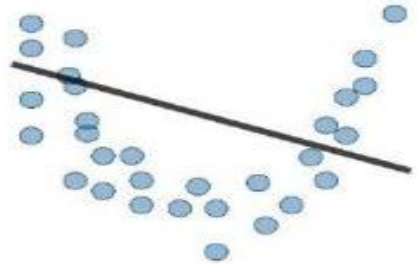
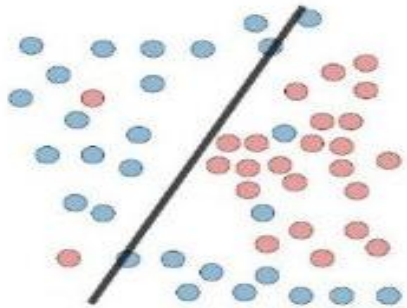
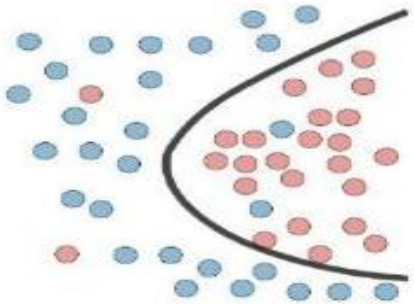
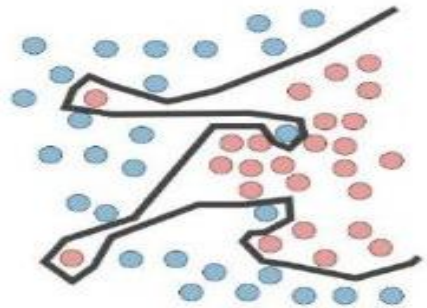
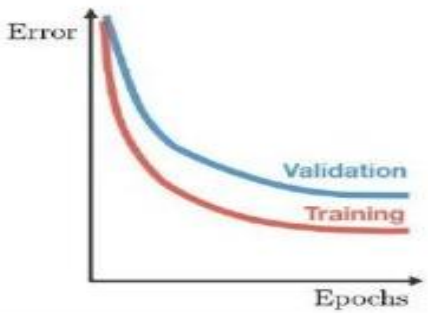
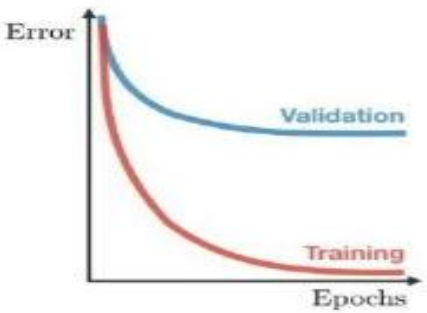


Overfitting & Underfitting

- Training a Model:
  - **Overfit**: if the model is performing much better on train set but not performing well on cross-validation set
  - **Overfit** is called as **“High Variance”** problem

  - **Underfit**: if the model is not performing well on train set itself
  - **Underfit** is considered as **“High Bias”** problem

MIT-WPU
॥ विश्वशान्तिर्ध्रुवं ध्रुवा ॥

	Underfitting	Just right	Overfitting
Symptoms	- High training error - Training error close to test error - High bias	- Training error slightly lower than test error	- Low training error - Training error much lower than test error - High variance
Regression			
Classification			
Deep learning			
Remedies	- Complexify model - Add more features - Train longer		- Regularize - Get more data